ADVANCE INFORMATION

Intel387™ SL MOBILE MATH COPROCESSOR

- **Static Intel387 Math CoProcessor Core**
  - **Lower Power Consumption for Portable PC's**
  - **½ Watt Active, 0.1 mW in Standby Mode**
- **Compatible with Intel386™ SX and Intel386 SL Microprocessors**
  - **Supports All Intel386 SL Power Management Modes**
- **Socket Compatible with Intel387 SX Math CoProcessor**
  - **Hardware and Software Compatible**
  - **Supported by Over 2100 Commercial Software ?ackages**
  - **Higher Performance Than Industry Standard Intel387 SX Math CoProcessor**
  - **10% to 25% Increase on Whetstone and Livermore Benchmarks**
- **Available in the Same 68-pin PLCC package as the Intel387 SX Math CoProcessor**

  See Intel Packaging Specification, Order #231369

3

The Intel387 SL Mobile Math CoProcessor is an extension of the Intel386 SL and SX Microprocessor architecture. This fully static device provides high performance floating point operations while maintaining low power consumption for the portable and desktop applications. The internal Power Management Unit reduces power consumption by 95% when the device is idle and the static core supports the automatic stop clock mode of the Intel386 SL Microprocessor, reducing current load to the 25 μA range. A new recognition feature provides simple programming of system power management circuitry.

The Intel387 SL Mobile Math CoProcessor is socket compatible with the Intel387 SX Math CoProcessor, packaged in a 68-pin PLCC package, and manufactured on Intel's advanced CHMOS IV Px48 technology.

290427-2

Intel386™ and Intel387™ are trademarks of Intel Corporation.

February 1992
Order Number: 290427-001

Intel387™ SL Mobile Math CoProcessor

CONTENTS PAGE

1.0 PIN ASSIGNMENT 3-5
1.1 Pin Description Table 3-6
2.0 FUNCTIONAL DESCRIPTION 3-7
2.1 Feature List 3-7
2.2 Math CoProcessor Architecture 3-7
2.3 Power Management 3-8
2.3.1 Dynamic Mode 3-8
2.3.2 Idle Mode 3-8
2.3.3 Standby Mode 3-8
2.4 Compatibility 3-8
2.5 Performance 3-9
3.0 PROGRAMMING INTERFACE 3-9
3.1 Instruction Set 3-9
3.1.1 Data Transfer Instructions 3-9
3.1.2 Arithmetic Instructions 3-10
3.1.3 Comparison Instructions 3-10
3.1.4 Transcendental Instructions 3-10
3.1.5 Load Constant Instructions ... 3-10
3.1.6 Processor Instructions 3-10
3.2 Register Set 3-11
3.2.1 Status Word (SW) Register ... 3-12
3.2.2 Control Word (CW) Register 3-15
3.2.3 Device Word (DW) Register 3-17
3.2.4 Signature Word (SG) Register 3-17
3.2.5 Data Register 3-17
3.2.6 Tag Word (TW) Register 3-18
3.2.7 Instruction and Data Pointers 3-19
3.3 Data Types 3-22
3.4 Interrupt Description 3-22
3.5 Exception Handling 3-23
3.6 Initialization 3-24
3.7 Processing Modes 3-24
3.8 Programming Support 3-24

CONTENTS PAGE

4.0 HARDWARE SYSTEM INTERFACE 3-24
4.1 Signal Description 3-25
4.1.1 Intel386 CPU Clock 2 (CPUCLK2) 3-25
4.1.2 Intel387 Math CoProcessor Clock 2 (NUMCLK2) 3-25
4.1.3 Clocking Mode (CKM) 3-26
4.1.4 System Reset (RESETIN) 3-26
4.1.5 Processor Request (PEREQ) 3-26
4.1.6 Busy Status (BUSY#) 3-26
4.1.7 Error Status (ERROR#) 3-26
4.1.8 Data Pins (D15–D0) 3-26
4.1.9 Write/Read Bus Cycle (W/R#) 3-26
4.1.10 Address Stobe (ADS#) 3-26
4.1.11 Bus Ready Input (READY#) 3-27
4.1.12 Ready Output (READYO#) 3-27
4.1.13 Status Enable (STEN) 3-27
4.1.14 Math CoProcessor Select 1 (NPS1#) 3-27
4.1.15 Math CoProcessor Select 2 (NPS2) 3-27
4.1.16 Command (CMD0#) 3-27
4.1.17 System Power (V_{CC}) 3-27
4.1.18 System Ground (V_{SS}) 3-27
4.2 System Configuration 3-27
4.2.1 Intel386 SX Microprocessor System 3-28
4.3 Math CoProcessor Architecture ... 3-29
4.3.1 Bus Control Logic 3-29
4.3.2 Data Interface and Control Unit 3-29
4.3.3 Floating Point Unit 3-29
4.3.4 Power Management Unit 3-29

CONTENTS

PAGE

4.4 Bus Cycles 3-29

4.4.1 Intel387 SL Mobile Math CoProcessor Addressing 3-30

4.4.2 CPU/Math CoProcessor Synchronization 3-30

4.4.3 Synchronous/Asynchronous Modes 3-30

4.4.4 Automatic Bus Cycle Termination 3-30

5.0 BUS OPERATION 3-30

5.1 Non-pipelined Bus Cycles 3-31

5.1.1 WRITE CYCLE 3-31

5.1.2 READ CYCLE 3-32

5.2 Pipelined Bus Cycles 3-32

5.3 Mixed Bus Cycles 3-33

5.4 BUSY# and PEREQ Timing Relationship 3-35

6.0 PACKAGE SPECIFICATIONS 3-36

6.1 Mechanical Specifications 3-36

6.2 Thermal Specifications 3-36

CONTENTS

PAGE

7.0 ELECTRICAL CHARACTERISTICS 3-36

7.1 Absolute Maximum Ratings 3-36

7.2 D.C. Characteristics 3-37

7.3 A.C. Characteristics 3-38

8.0 Intel387 SL MOBILE MATH COPROCESSOR INSTRUCTION SET 3-45

9.0 REVISION HISTORY 3-48

APPENDIX A—PROGRAMMER'S REFERENCE MANUAL UPDATES ... 3-49

APPENDIX B—Intel387 SL MOBILE MATH COPROCESSOR COMPATIBILITY 3-51

B.1 Intel387 SX Compatibility 3-51

B.2 8087/80287 Compatibility 3-51

B.2.1 General Differences 3-51

B.2.2 Exceptions 3-52

APPENDIX C—COMPATIBILITY BETWEEN THE 80287 AND 8087 MATH COPROCESSOR 3-53

CONTENTS PAGE

FIGURES

Figure 1-1 Intel387 SL Mobile Math CoProcessor Pinout 3-5

Figure 2-1 Intel387 SL Mobile Math CoProcessor Block Diagram 3-8

Figure 3-1 Intel 386 SL CPU and Intel387 Math CoProcessor Register Set 3-11

Figure 3-2 Status Word 3-12

Figure 3-3 Control Word 3-16

Figure 3-4 Device Word Register 3-17

Figure 3-5 Signature Word Register ... 3-17

Figure 3-6 Tag Word Register 3-18

Figure 3-7 Instruction and Data Pointer Image in Memory, 32-Bit Protected Mode Format 3-19

Figure 3-8 Instruction and Data Pointer Image in Memory, 16-Bit Protected Mode Format 3-20

Figure 3-9 Instruction and Data Pointer Image in Memory, 32-Bit Real Mode Format 3-20

Figure 3-10 Instruction and Data Pointer Image in Memory, 16-Bit Real Mode Format 3-20

Figure 4-1 Intel386 SL CPU Math CoProcessor Interface 3-28

Figure 5-1 Bus State Diagram 3-31

Figure 5-2 Non-Pipelined Read and Write Cycles 3-32

Figure 5-3 Fastest Transition to and from Pipelined Cycles 3-33

Figure 5-4 Pipelined Cycles with Wait States 3-34

Figure 5-5 BUSY#, PEREQ, and ERROR# Timing Relationships 3-35

Figure 7-1a Typical Output Valid Delay vs Load Capacitance at Max Operating Temperature 3-40

Figure 7-1b Typical Output Slew Time vs Load Capacitance at Max Operating Temperature 3-40

Figure 7-1c Maximum I_{CC} vs Frequency 3-41

CONTENTS PAGE

Figure 7-2 CPUCLK2/NUMCLK2 Waveform and Measurement Points for Input/Output 3-41

Figure 7-3 Output Signals 3-42

Figure 7-4 Input and I/O Signals 3-42

Figure 7-5 RESET Signal 3-43

Figure 7-6 Float from STEN 3-43

Figure 7-7 Other Parameters 3-44

TABLES

Table 1-1 Pin Cross Reference—Functional Grouping 3-5

Table 3-1 Condition Code Interpretation 3-13

Table 3-2 Condition Code Interpretation after FPREM and FPREM1 Instructions 3-14

Table 3-3 Condition Code Resulting from Comparison 3-14

Table 3-4 Condition Code Defining Operand Class 3-14

Table 3-5 Mapping Condition Codes to Intel386 CPU Flag Bits 3-15

Table 3-6 Intel387 SL Mobile Math CoProcessor Data Type Representation in Memory .. 3-21

Table 3-7 CPU Interrupt Vectors Reserve for Math CoProcessor 3-22

Table 3-8 Intel387 SL Mobile Math CoProcessor Exceptions 3-23

Table 4-1 Pin Summary 3-25

Table 4-2 Output Pin Status during Reset 3-26

Table 4-3 Bus Cycle Definition 3-29

Table 6-1 Thermal Resistances (°C/Watt) θ_{JC} and θ_{JA} 3-36

Table 6-2 Maximum T_A at Various Airflows 3-36

Table 7-1 D.C. Specifications 3-37

Table 7-2a Timing Requirements of the Bus Interface Unit 3-38

Table 7-2b Timing Requirements of the Execution Unit 3-39

Table 7-2c Other AC Parameters 3-39

Table 8-1 Instruction Formats 3-45

1.0 PIN ASSIGNMENT

The Intel387 SL Mobile Math CoProcessor pinout as viewed from the top side of the component is shown in Figure 1-1. V_{CC} and V_{SS} (GND) connections must be made to multiple pins. The circuit board should include V_{CC} and V_{SS} planes for power distribution and all V_{CC} and V_{SS} pins must be connected to the appropriate plane.

NOTE:
Pins identified as N.C. should remain completely unconnected.

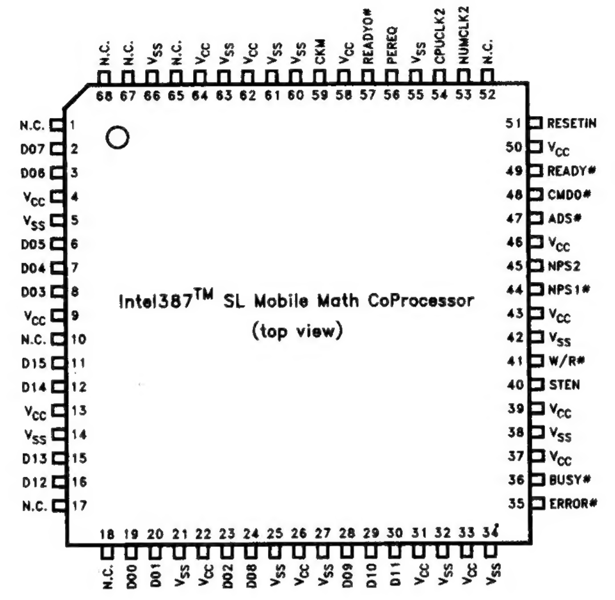


Figure 1-1. Intel387 SL Mobile Math CoProcessor Pinout

Table 1-1. Pin Cross Reference—Functional Grouping

BUSY#	36	D00	19	V_{CC}	4	V_{SS}	5	N.C.	1
PEREQ	56	D01	20		9		14		10
ERROR#	35	D02	23		13		21		17
ADS#	47	D03	8		22		25		18
CMD0#	48	D04	7		26		27		52
NPS1#	44	D05	6		31		32		65
NPS2	45	D06	3		33		34		67
STEN	40	D07	2		37		38		68
W/R#	41	D08	24		39		42		
READY#	49	D09	28		43		55		
READYO#	57	D10	29		46		60		
		D11	30		50		61		
		D12	16		58		63		
CKM	59	D13	15		62		66		
CPUCLK2	54	D14	12		64				
NUMCLK2	53	D15	11						
RESETIN	51								

1.1 Pin Description Table

The following table lists a brief description of each pin on the Intel387 SL Mobile Math CoProcessor. For a more complete description refer to Section 4.1 Signal Description. The following definitions are used in these descriptions:

#	The signal is active LOW.
I	Input Signal
O	Output Signal
I/O	Input and Output Signal

Symbol	Type	Name and Function
ADS#	I	**ADDRESS STROBE** indicates that the address and bus cycle definition is valid.
BUSY#	O	**BUSY** indicates that the Math CoProcessor is currently executing an instruction.
CKM	I	**CLOCKING MODE** is used to select synchronous or asynchronous clock modes.
CMD0	I	**COMMAND** determines whether an opcode or operand are being sent to the Math CoProcessor. During a read cycle it indicates which register group is being read.
CPUCLK2	I	**CPU CLOCK** input provides the timing for the bus interface unit and the execution unit in synchronous mode.
D15–D0	I/O	**DATA BUS** is used to transfer instructions and data between the Math CoProcessor and CPU.
ERROR#	O	**ERROR** signals that an unmasked exception has occurred.
NC	—	**NO CONNECT** should always remain unconnected. Connection of a N.C. pin may cause the Math CoProcessor to malfunction or be incompatible with future steppings.
NPS1#	I	**NPX SELECT 1** is used to select the Math CoProcessor.
NPS2	I	**NPX SELECT 2** is used to select the Math CoProcessor.
NUMCLK2	I	**NUMERICS CLOCK** is used in asynchronous mode to drive the Floating Point Execution Unit.
PEREQ	O	**PROCESSOR EXTENSION REQUEST** signals the CPU that the Math CoProcessor is ready for data transfer to/from its FIFO.
READY#	I	**READY** indicates that the bus cycle is being terminated.
READYO#	O	**READY OUT** signals the CPU that the Math CoProcessor is terminating the bus cycle.
RESETIN	I	**SYSTEM RESET** terminates any operation in progress and forces the Math CoProcessor to enter a dormant state.
STEN	I	**STATUS ENABLE** serves as a master chip select for the Math CoProcessor. When inactive, this pin forces all outputs and bi-directional pins into a floating state.
W/R#	I	**WRITE/READ** indicates whether the CPU bus cycle in progress is a read or a write cycle.
V_{CC}	I	**SYSTEM POWER** provides the +5V nominal D.C. supply input.
V_{SS}	I	**SYSTEM GROUND** provides the 0V connection from which all inputs and outputs are measured.

2.0 FUNCTIONAL DESCRIPTION

The Intel387 SL Mobile Math CoProcessor is designed to support the Intel386™ SL and Intel386 SX Microprocessors and effectively extends the CPU architecture by providing fast execution of arithmetic instructions and transcendental functions. This fully static component contains internal power management circuitry for reduced active power dissipation and an automatic idle mode. The Intel387 SL Mobile Math CoProcessor supports the stop clock mode of the Intel386 SL CPU. Although designed primarily for the notebook and portable computer market, the Intel387 SL Mobile Math CoProcessor is fully compatible with the Intel387 SX Mobile Math CoProcessor and can be used in any system designed to accept an Intel387 SX Math CoProcessor.

2.1 Feature List

- New Static design provides low power dissipation in active and idle modes.
- Two new registers provide recognition of device features such as static and steppings.
- Higher Performance, 10%–25% higher benchmark performance than the Intel387 SX Math CoProcessor.
- High Performance 84-bit Internal Architecture
- Eight 80-bit Numeric Registers, usable as individually addressable general registers or as a register stack.
- Full-range transcendental operations for SINE, COSINE, TANGENT, ARCTANGENT, and LOGARITHM.
- Programmable rounding modes and notification of rounding effects.
- Exception reporting either by software polling or hardware interrupts.
- Fully compatible with the Intel386 SL and SX Microprocessors.
- Expands Intel386 SX CPU data types to include 32-bit, 64-bit, and 80-bit Floating Point; 32-bit and 64-bit Integers; and 18 Digit BCD Operands.
- Directly extends the Intel386 SX and Intel386 SL CPU Instruction Set to trigonometric, logarithmic, exponential, and arithmetic functions for all data types.
- Operates independently of Real, Protected, and Virtual-86 Modes of the Intel386 SX and SL Microprocessors.
- Fully compatible with the Intel387 SX and DX Math CoProcessors. Implements all Intel387 Math CoProcessor architectural enhancements over 8087 and 80287.
- Implements ANSI/IEEE Standard 754-1985 for binary floating point arithmetic.
- Upward Object Code compatible from 8087 and 80287.

2.2 Math CoProcessor Architecture

As shown in Figure 2-1, the Intel387 SL Mobile Math CoProcessor is internally divided into four sections; the Bus Control Logic, the Data Interface and Control Logic, the Floating Point Unit, and the Power Management Unit. The Bus Control Logic is responsible for the CPU bus tracking and interface. The Data Interface and Control Unit latches data and decodes instructions. The Floating Point Unit executes the mathematical instructions. The Power Management Unit is new to the Intel387 family and is the nucleus of the static architecture. It is responsible for shutting down idle sections of the device to save power.

Microprocessor/Math CoProcessor Interface
The Intel386 CPU interprets the pattern 11011B in most significant five bits of an instruction as an opcode intended for a math coprocessor. Instructions thus marked are called ESCAPE or ESC instructions. Upon decoding the instruction as an ESC instruction, the Intel386 CPU transfers the opcode to the math coprocessor through an I/O write cycle at a dedicated address (8000F8H) outside the normal programmed I/O address range. The math coprocessor has dedicated output signals for controlling the data transfer and notifying the CPU if the Math CoProcessor is busy or that a floating point error has occurred.

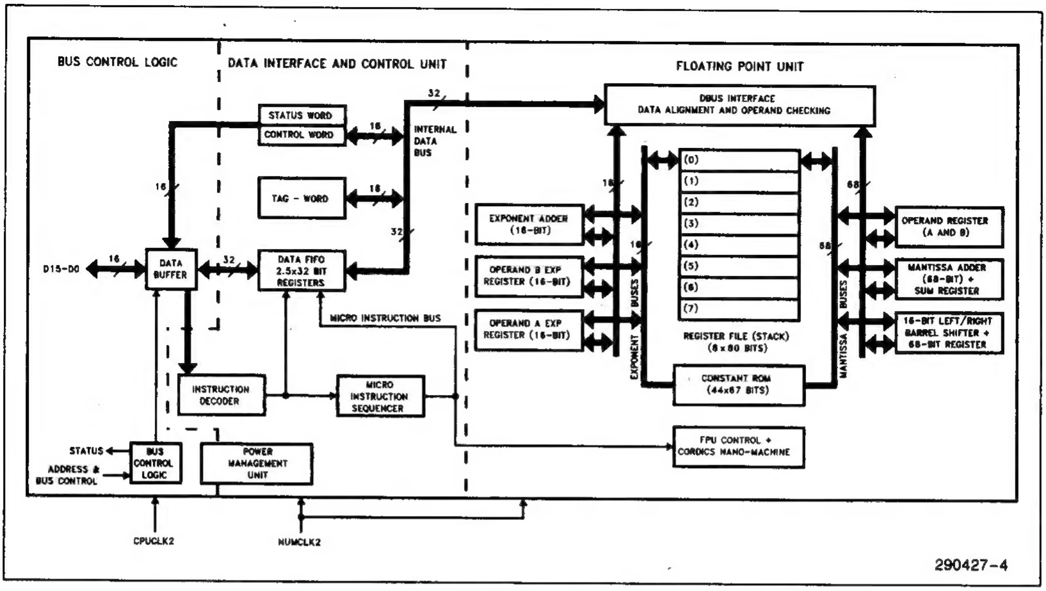

BUS CONTROL LOGIC
DATA INTERFACE AND CONTROL UNIT
FLOATING POINT UNIT
32
STATUS WORD
CONTROL WORD
16
INTERNAL DATA BUS
16
TAG - WORD
16
32
D15-D0
16
DATA BUFFER
32
DATA FIFO 2.5x32 BIT REGISTERS
MICRO INSTRUCTION BUS
INSTRUCTION DECODER
MICRO INSTRUCTION SEQUENCER
STATUS
ADDRESS & BUS CONTROL
BUS CONTROL LOGIC
POWER MANAGEMENT UNIT
CPUCLK2
NUMCLK2
DBUS INTERFACE
DATA ALIGNMENT AND OPERAND CHECKING
16
68
EXPONENT ADDER (16-BIT)
OPERAND B EXP REGISTER (16-BIT)
OPERAND A EXP REGISTER (16-BIT)
16
68
EXPONENT BUSES
MANTISSA BUSES
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
REGISTER FILE (STACK) (8 x 80 BITS)
CONSTANT ROM (44x67 BITS)
FPU CONTROL + CORDICS NANO-MACHINE
OPERAND REGISTER (A AND B)
MANTISSA ADDER (68-BIT) + SUM REGISTER
16-BIT LEFT/RIGHT BARREL SHIFTER + 68-BIT REGISTER
290427-4

Figure 2-1. Intel387 SL Mobile Math CoProcessor Block Diagram

2.3 Power Management

The Intel387 SL Mobile Math CoProcessor offers three modes of power management; dynamic, idle, and standby.

2.3.1 DYNAMIC MODE

Dynamic Mode is when the device is executing an instruction. Using Intel's CHMOS IV technology, the Intel387 SL Mobile Math CoProcessor draws considerably less power than its predecessor. The active power supply current is reduced to approximately 100 mA at 20 MHz and provides low case temperatures.

2.3.2 IDLE MODE

When an instruction is not being executed, the Intel387 SL Mobile Math CoProcessor will automatically change to ***Idle Mode***. Three clocks after completion of the previous instruction, the internal power manager shuts down the floating point execution unit and all non-essential circuitry. Only portions of the Bus Interface Unit remain active to monitor the CPU bus activity and to accept the next instruction when it is transferred. When the CPU transfers the next instruction to the Math CoProcessor, the Intel387 SL Mobile Math CoProcessor accepts the instruction and ramps the internal core within one clock so there is no impact to performance or throughput. In idle mode, the Intel387 SL Mobile Math CoProcessor draws typically 4 mA of current and reduces case temperature to near ambient.

2.3.3 STANDBY MODE

The Intel387 SL Mobile Math CoProcessor also supports the Intel386 SL Microprocessor's programmable slow and stop clock methods. In ***Standby Mode***, the CPU monitors the Math CoProcessor BUSY# output and if the device is not active for a predefined period the CPU slows or stops the Math CoProcessor clock. With the clock stopped the Intel387 SL Mobile Math CoProcessor will draw approximately 25 μA of current while retaining register and data integrity.

Although the Intel386 SL CPU supports slow/stop clock modes only when the Math CoProcessor is not active, the Intel387 SL Mobile Math CoProcessor clock can be stopped at any time, including while executing an instruction. However, to minimize current the Math CoProcessor should not be driving the data bus when the clock is stopped. The clock can be resumed at any time provided that it is synchronized with the bus activity and in phase with the CPU clock.

NOTE:
In asynchronous clock mode (CKM = 0), the internal idle mode is disabled and external slow or stop clock modes are not guaranteed.

2.4 Compatibility

The Intel387 SL Mobile Math CoProcessor is fully compatible with both the Intel386 SX and Intel386 SL Microprocessors and supports all of the Intel386 SL Microprocessor Power Management modes.

The Intel387 SL Mobile Math CoProcessor is fully compatible with the Intel387 SX Math CoProcessor. The Intel387 SL Mobile Math CoProcessor can be plugged into any socket designed for the Intel387 SX Math CoProcessor with no change to software or hardware. Due to the increased performance and internal pipelining effects, diagnostic programs should never use instruction execution time for test purposes.

2.5 Performance

The increased performance of floating point calculations can be attributed to the 84-bit architecture and floating point processor. For the CPU to execute floating point calculations requires very long software emulation methods with reduced resolution and accuracy. The performance of the Intel387 SL Mobile Math CoProcessor has been further enhanced through improvements in the internal microcode and through internal architectural changes. These refinements will increase Whetstone benchmarks by approximately 10% to 25% over the Intel387 SX Math CoProcessor.

Real performance, however, should be measured with application software. Depending upon software coding, system overhead, and percentage of floating point instructions, performance can vary significantly.

3.0 PROGRAMMING INTERFACE

The Intel387 SL Mobile Math CoProcessor effectively extends to an Intel386 Microprocessor system additional instructions, registers, data types, and interrupts specifically designed to facilitate high-speed floating point processing. All communication between the CPU and the Math CoProcessor is transparent to applications software. The CPU automatically controls the Math CoProcessor whenever a numerics instruction is executed. All physical memory and virtual memory of the CPU are available for storage of the instructions and operands of programs that use the Math CoProcessor. All memory addressing modes, including use of displacement, base register, index register, and scaling are available for addressing numerical operands.

The Intel387 SL Mobile Math CoProcessor is software compatible with the Intel387 SX and DX Math CoProcessors and supports all applications written for the Intel386 CPU and Intel387 Math CoProcessors.

The Intel387 SL Mobile Math CoProcessor has additional features to assist the system programmer. The Intel387 SL Mobile Math CoProcessor includes a new Device Word register for quick and easy determination of it's static capability to assist in the programming of system level power managers. Also included is a Signature Word register which can be used to determine the current version of the Intel387 SL Mobile Math CoProcessor.

3.1 Instruction Set

The Intel386 CPU interprets the pattern 11011B in most significant five bits of an instruction as an opcode intended for a math coprocessor. Instructions thus marked are called ESCAPE or ESC instruction.

The typical Math CoProcessor instruction accepts one or two operands and produces one or sometimes two results. In two-operand instructions, one operand is the contents of the Math CoProcessor register, while the other may be a memory location. The operands of some instructions are predefined; for example, FSQRT always takes the square root of the number in the top stack element.

The Intel387 SL Mobile Math CoProcessor instruction set can be divided into six groups. The following sections gives a brief description of each instruction. Section 8.0 defines the instruction format and byte fields. Further details can be obtained from the Intel387 User's Manual, Programmer's Reference, Order #231917.

3.1.1 DATA TRANSFER INSTRUCTIONS

The class includes the operations that load, store, and convert operands of any support data types.

Real Transfers

FLD	Load Real (single, double, extended)
FST	Store Real (single, double)
FSTP	Store Real and pop (single, double, extended)
FXCH	Exchange registers

Integer Transfers

FILD	Load (convert from) Integer (word, short, long)
FIST	Store (convert to) Integer (word, short)
FISTP	Store (convert to) Integer and pop (word, short, long)

Packed Decimal Transfers

FBLD	Load (convert from) packed decimal
FBSTP	Store packed decimal and pop

3.1.2 ARITHMETIC INSTRUCTIONS

This class of instructions provide variations on the basic add, subtract, multiply, and divide operations and a number of other basic arithmetic operations. Operands may reside in registers or one operand may reside in memory.

Addition

FADD	Add Real
FADDP	Add Real and pop
FIADD	Add Integer

Subtraction

FSUB	Subtract Real
FSUBP	Subtract Real and pop
FISUB	Subtract Integer
FSUBR	Subtract Real reversed
FSUBRP	Subtract Real reversed and pop
FISUBR	Subtract Integer reversed

Multiplication

FMUL	Multiply Real
FMULP	Multiply Real and pop
FIMUL	Multiply Integer

Division

FDIV	Divide Real
FDIVP	Divide Real and pop
FIDIV	Divide Integer
FDIVR	Divide Real reversed
FDIVRP	Divide Real reversed and pop
FIDIVR	Divide Integer reversed

Other Operations

FSQRT	Square Root
FSCALE	Scale
FPREM	Partial Remainder
FPREM1	IEEE standard partial remainder
FRNDINT	Round to Integer
FXTRACT	Extract Exponent and Significand
FABS	Absolute Value
FCHS	Change sign

3.1.3 COMPARISON INSTRUCTION

Instructions of this class allow comparison of numbers of all supported real and integer data types. Each of these instructions analyzes the top stack element often in relationship to another operand and reports the result as a condition code in the status word.

FCOM	Compare Real
FCOMP	Compare Real and pop
FCOMPP	Compare Real and pop twice
FUCOM	Unordered compare Real
FUCOMP	Unordered compare Real and pop
FUCOMPP	Unordered compare Real and pop twice
FICOM	Compare Integer
FICOMP	Compare Integer and pop
FTST	Test
FXAM	Examine

3.1.4 TRANSCENDENTAL INSTRUCTIONS

This group of the Intel387 operations includes trigonometric, inverse trigonometric, logarithmic and exponential functions. The transcendental operate on the top one or two stack elements, and they return their results to the stack. The trigonometric operations assume their arguments are expressed in radians. The logarithmic and exponential operations work in base 2.

FSIN	Sine
FCOS	Cosine
FSINCOS	Sine and cosine
FPTAN	Tangent
FPATAN	Arctangent of ST(1)/ST
F2XM1	$2^x - 1$
FYL2X	$Y * \log_2 X$
FYL2XP1	$Y * \log_2(X + 1)$

3.1.5 LOAD CONSTANT INSTRUCTIONS

Each of these instructions loads (pushes) a commonly used constant onto the stack. The constants have extended real values nearest to the infinitely precise numbers. The only error that can be generated is an Invalid Exception if a stack overflow occurs.

FLDZ	Load +0.0
FLD1	Load +1.0
FLDPI	Load π
FLDL2T	Load $\log_2 10$
FLDL2E	Load $\log_2 e$
FLDLG2	Load $\log_{10} 2$
FLDLN2	Load $\log_e 2$

3.1.6 PROCESSOR INSTRUCTIONS (ADMINISTRATIVE)

FINIT	Initialize Math CoProcessor
FLDCW	Load Control Word

FSTCW	Store Control Word
FLDCW	Load Status Word
FSTSW	Store Status Word
FSTSW AX	Store Status Word to AX register
FSTDW AX	Store Device Word to AX register
FSTSG AX	Store Signature Word to AX register
FCLEX	Clear Exceptions
FSTENV	Store Environment
FLDENV	Load Environment
FSAVE	Save State
FRSTOR	Restore State

FINCSTP	Increment Stack pointer
FDECSTP	Decrement Stack pointer
FFREE	Free Register
FNOP	No Operation
FWAIT	Report Math CoProcessor Error

3.2 Register Set

Figure 3-1 shows the Intel387 SL Mobile Math CoProcessor register set. When a Math CoProcessor is present in a system, programmers may use these registers in addition to the registers normally available on the CPU.

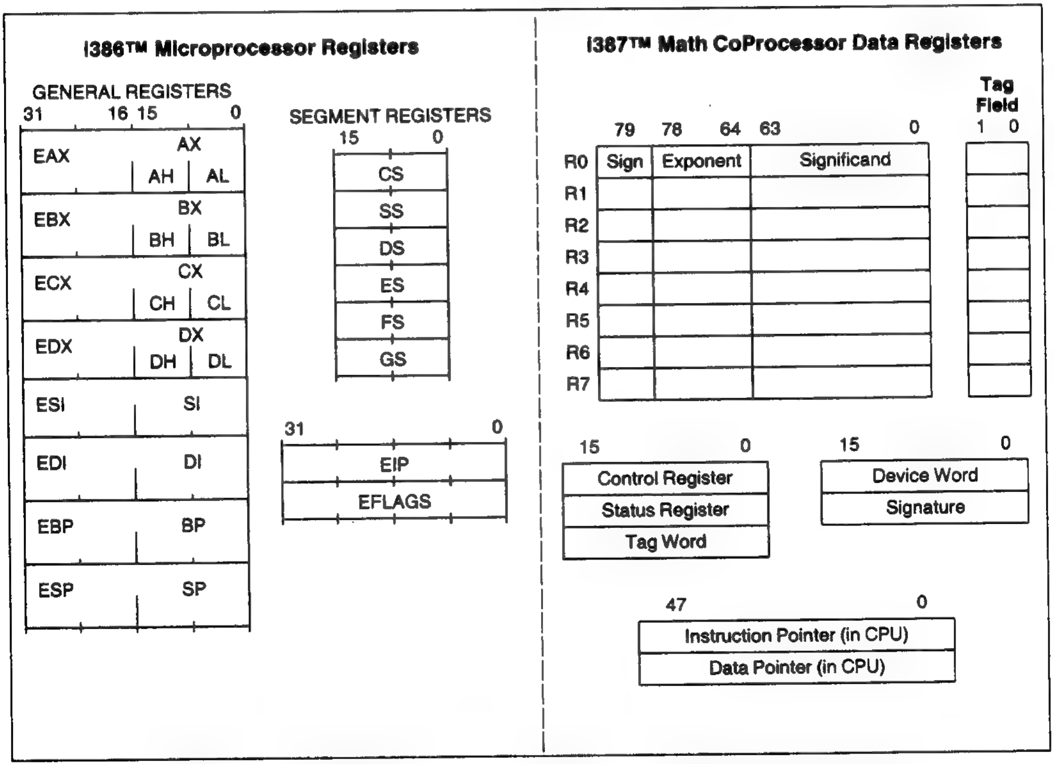
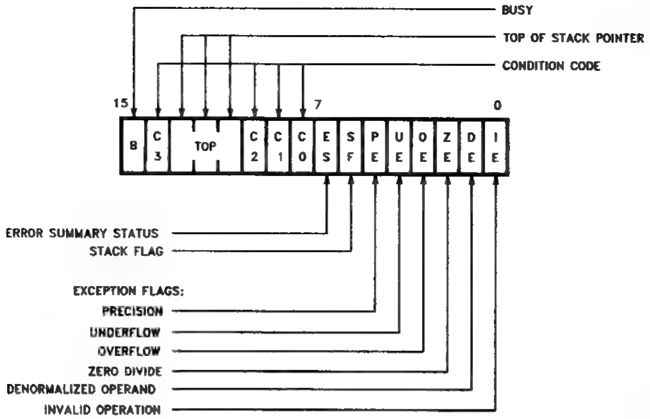


Figure 3-1. Intel386 SL CPU and Intel387 Math CoProcessor Register Set



290427-5

ES is set if any unmasked exception bit is set; cleared otherwise. See Table 2-2 for interpretation of condition code.
TOP values:
000 = Register 0 is Top of Stack
001 = Register 1 is Top of Stack
.
.
.
111 = Register 7 is Top of Stack
For definitions of exceptions, refer to the section entitled "Exception Handling"

Figure 3-2. Status Word

3.2.1 STATUS WORD (SW) REGISTER

The 16-bit status word (in the status register) shown in Figure 3-2 reflects the overall state of the Math CoProcessor. It can be read and inspected by programs using the FSTSW memory or FSTSW AX instructions.

Bit 15, the Busy bit (B) is included for 8087 compatibility only. It always has the same value as the Error Summary bit (ES, bit 7 of status word); it does not indicate the status of the BUSY# output of the Math CoProcessor.

Bits 13–11 (TOP) serves as the pointer to the Math CoProcessor data register that is the current Top-Of-Stack. The significance of the stack top is described in Section 3.2.5 Data Registers.

The four numeric condition code bits (C_3–C_0, Bit 14, 10–8) are similar to the flags in a CPU; instructions that perform arithmetic operations update these bits to reflect the outcome. The effects of the instructions on the condition code are summarized in Tables 3-1 through 3-4. These condition code bits are used principally for conditional branching. The FSTSW AX instructions stores the Math CoProcessor status word directly to the CPU AX register, allowing the condition codes to be inspected efficiently by Intel386 CPU code. The Intel386 CPU SAHF instruction can copy C_3–C_0 directly to the flag bits to simplify conditional branching. Table 3-5 shows the mapping of these bits to the Intel386 CPU flag bits.

Bit 7 is the error summary (ES) status bit. This bit is set if any unmasked exception bit is set; it is clear otherwise. If this bit is set, the ERROR# signal is asserted.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow from other kinds of invalid operations. When SF is set, bit 9 (C_1) distinguishes between stack overflow ($C_1 = 1$) or underflow ($C_1 = 0$).

Bit 5–0 are the six exception flags of the status word and are set to indicate that during an instruction execution the Math CoProcessor has detected one of six possible exception conditions since these status bits were last cleared or reset. Section 3.5 entitled Exception Handling explains how they are set and used.

The exception flags are "sticky" bits and can only be cleared by the instructions FINIT, FCLEX, FLDENV, FSAVE, and FRSTOR. Note that when a new value is loaded into the status word by the FLDENV or FRSTOR instruction, the value of ES (bit 7) and B (bit 15) are not derived from the values loaded from memory but rather are dependent upon the values of the exception flags (bits 5–0) in the status word and their corresponding masks in the control word. If ES is set in such a case, the ERROR# output of the Math CoProcessor is activated immediately.

Table 3-1. Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 3-2)	Three least significant bits of quotient Q2	Q0	Q1 or O/U#	Reduction 0 = complete 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 3-3)		Zero or O/U#	Operand is not comparable (Table 3-3)
FXAM	Operand class (see Table 3-4)		Sign or O/U#	Operand class (Table 3-4)
FCHS, FABS, FXCH, FINCSTP, FDECSTP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FINIT, FSAVE	UNDEFINED			

O/U# When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).

Reduction If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.

Roundup When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.

UNDEFINED Do not rely on finding any specific value in these bits.

Table 3-2. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further interation required for complete reduction	
0	Q1	Q0	Q2	Q MOD8	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
	0	0	0	0	
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
	1	1	1	7	

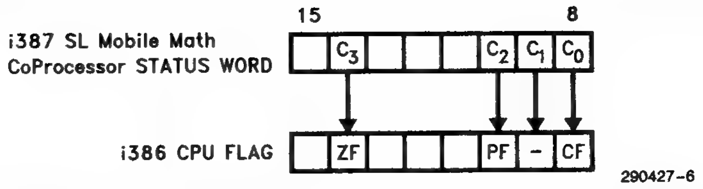
Table 3-3. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 3-4. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	− Unsupported
0	0	1	1	− NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	− Normal
0	1	1	1	− Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	− 0
1	0	1	1	− Empty
1	1	0	0	+ Denormal
1	1	1	0	− Denormal

Table 3-5 Mapping Condition Codes to Intel386™ CPU Flag Bits



3.2.2 CONTROL WORD (CW) REGISTER

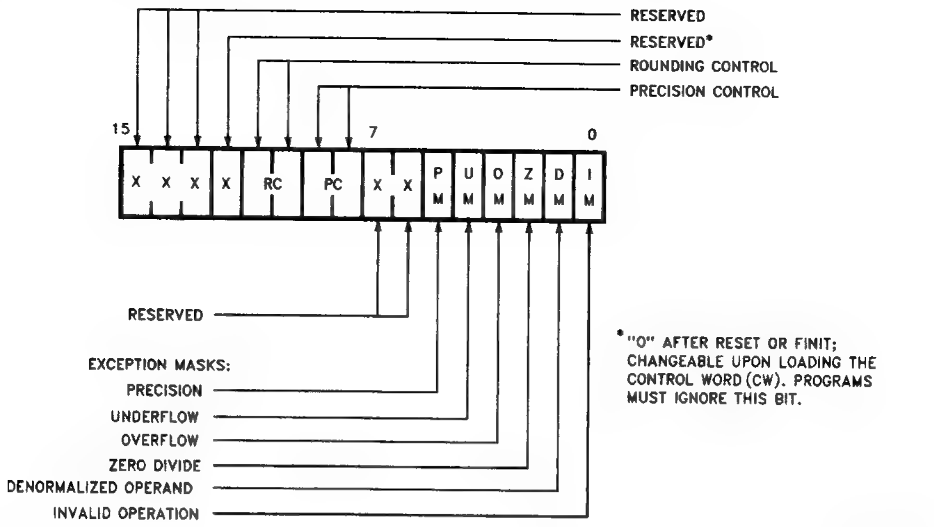
The Math CoProcessor provides the programmer with several processing options that are selected by loading a control word from memory into the control register. Figure 3-3 show the format and encoding of fields in the control word.

The low-order byte of the control word register is used to configure the exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the Math CoProcessor recognizes. See Section 3.5, Exception Handling, for further explanation on the exception control and definition.

The high-order byte of the control word is used to configure the Math CoProcessor operating mode, including precision, rounding and infinity control.

- The rounding control (RC) field (bits 11–10) provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS, and FCHS) and all transcendental instructions.
- The precision control (PC) field (bits 9–8) can be used to set the Math CoProcessor internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions FADD, FSUB(R), FMUL, FDIV(R), and FSQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.
- The "infinity control bit" (bit 12) is not meaningful to the Intel387 SL Mobile Math CoProcessor and programs must ignore its value. To maintain compatibility with the 8087 and 80287 (non-387 core), this bit can be programmed, however, regardless of its value the Intel387 SL Mobile Math CoProcessor always treats infinity in the affine sense ($-\infty < +\infty$). This bit is initialized to zero both after a hardware reset and after FINIT instruction.

All other bits are reserved and should not be programmed, to assure compatibility with future processors.



*"0" AFTER RESET OR FINIT; CHANGEABLE UPON LOADING THE CONTROL WORD (CW). PROGRAMS MUST IGNORE THIS BIT.

Precision Control
00—24 bits (single precision)
01—(reserved)
10—53 bits (double precision)
11—64 bits (extended precision)

Rounding Control
00—Round to nearest or even
01—Round down (toward $-\infty$)
10—Round up (toward $+\infty$)
11—Chop (truncate toward zero)

Figure 3-3. Control Word

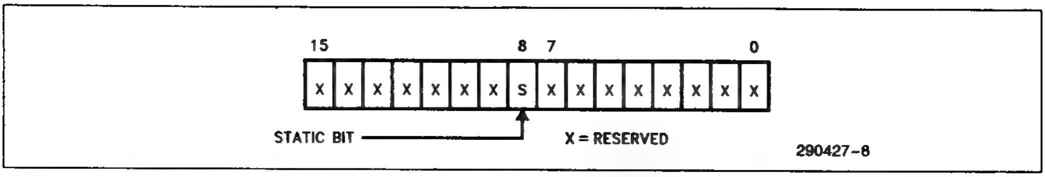


Figure 3-4. Device Word Register

3.2.3 DEVICE WORD (DW) REGISTER

This register is new to the Intel387 Math CoProcessors. The 16-bit device word register, shown in Figure 3-4, reflects available device specific options. This read-only register may be transferred to the CPU AX register using the FSTDW AX instruction (opcode DF E1), allowing inspection by firmware for system set up and initialization. This register is valid only after FINIT and before the status word has been changed to maintain compatibility with previous devices.

The static bit (S, bit 8), if set to a 1, indicates that the device is a static design which will allow the clock to be stopped without the loss of data integrity. See Appendix A for a programming example using the static bit of the Device Word register to set up the Intel386 SL CPU power management registers.

Bits 15–9 and bits 7–0 of the device word register are currently reserved. Software should not rely upon the state of these register locations.

See Appendix B for the Programmer's Reference Manual revision for the FSTDW AX opcode.

3.2.4 SIGNATURE WORD (SG) REGISTER

The 16-bit signature register provides the programmer with the Intel387 SL Mobile Math CoProcessor ID. Figure 3-5 reflects the register definition for the Intel387 SL Mobile Math CoProcessor A-0 step. This read-only register can be transferred to the CPU AX register using the FSTSG AX instruction (opcode DF E2). The signature register has three fields: the stepping revision, family, and version fields.

The stepping revision field (bits 7–0) indicates the major (bits 7–4) and minor (bits 3–0) stepping of the device.

The product family field (bits 11–8) indicates what family the Math CoProcessor belongs to. This field is defined as 03H to indicate the Intel387 family of math coprocessors.

The family version field (bits 15–12) indicates which member of the family of Intel math coprocessor products this Math CoProcessor belongs to. This field is defined as 02h to indicate the SL version of the Intel387 Math CoProcessor.

See Appendix B for the Programmer's Reference Manual revision for the FSTSG AX opcode.

3.2.5 DATA REGISTER

Intel387 SL Mobile Math CoProcessor data register set consist of eight registers (R0–R7) which are treated as both a stack and a general register file. Each of these data registers in the Math CoProcessor is 80 bits wide and is divided into fields corre-

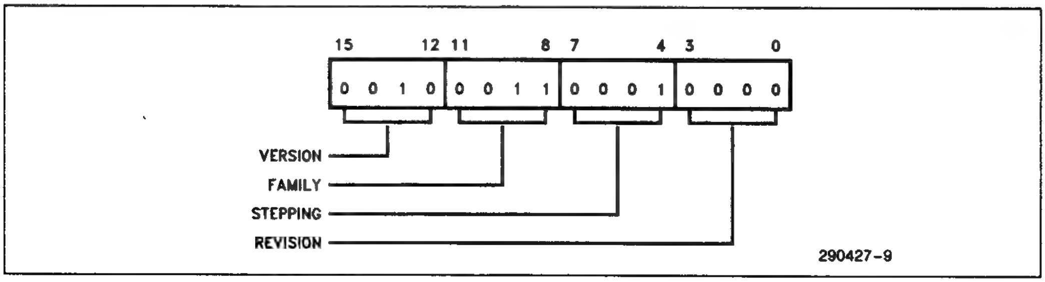


Figure 3-5. Signature Word Register

sponding to the Math CoProcessor's extended-precision real data type, which is used for internal calculations.

The Math CoProcessor register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as individually addressable registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "store and pop" operation stores the value from the current top register into memory and then increments TOP by one. The Math CoProcessor register stack grows "down" toward lower-addressed registers.

Most of the Intel387 Math CoProcessor operations use the register stack as the operand(s) and/or as a place to store the result. Instructions may address the data register either implicitly or explicitly. Many instructions operate on the register at the top of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. Explicit register addressing is also relative to TOP (where ST denotes the current stack top and ST(i) refers to the i'th register from the ST in the stack so the real register address in computed as ST+i).

3.2.6 TAG WORD (TW) REGISTER

The tag word marks the content of each numeric data register, as Figure 3-6 shows. Each two-bit tag represents one of the eight data register. The principal function of the tag word is to optimize the Math CoProcessor's performance and stack handling by making it possible to distinguish between empty and non-empty register locations. It also enables exception handlers to identify special values (e.g. NaNs or denormals) in the contents of a stack location without the need to perform complex decoding of the actual data.

15							0
TAG (7)	TAG (6)	TAG (5)	TAG (4)	TAG (3)	TAG (2)	TAG (1)	TAG (0)

NOTE:
The index i of tag(i) is not top-relative. A program typically uses the "top" field of Status Word to determine which tag(i) field refers to logical top of stack.
TAG VALUES:
00 = Valid
01 = Zero
10 = QNaN, SNaN, Infinity, Denormal and Unsupported Formats
11 = Empty

Figure 3-6. Tag Word Register

3.2.7 INSTRUCTION AND DATA POINTERS

Because the Math CoProcessor operates in parallel with the CPU, any exceptions detected by the Math CoProcessor may be reported after the CPU has executed the ESC instruction which caused. it. To allow identification of the numeric instruction which caused the exception, the Intel386 Microprocessor contains registers that aid in diagnosis. These registers supply the address of the failing instruction and the address of its numeric memory operand (if appropriate).

The instruction and data pointers are provided for user-written exception handlers. These registers are located in the CPU, but appear to be located in the Math CoProcessor because they are accessed by the ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR; which transfer the values between the registers and memory. Whenever the CPU executes a new ESC instruction (except administrative instructions), it saves the address of the instruction (including any prefixes that may be present), the address of the operand (if present) and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the CPU (protected mode or real-address mode) and depending on the operand size attribute in effect (32-bit operand or 16-bit operand). (See Figures 3-7, 3-8, 3-9, and 3-10.) Note that the value of the data pointer is *undefined* if the prior ESC instruction did not have a memory operand.

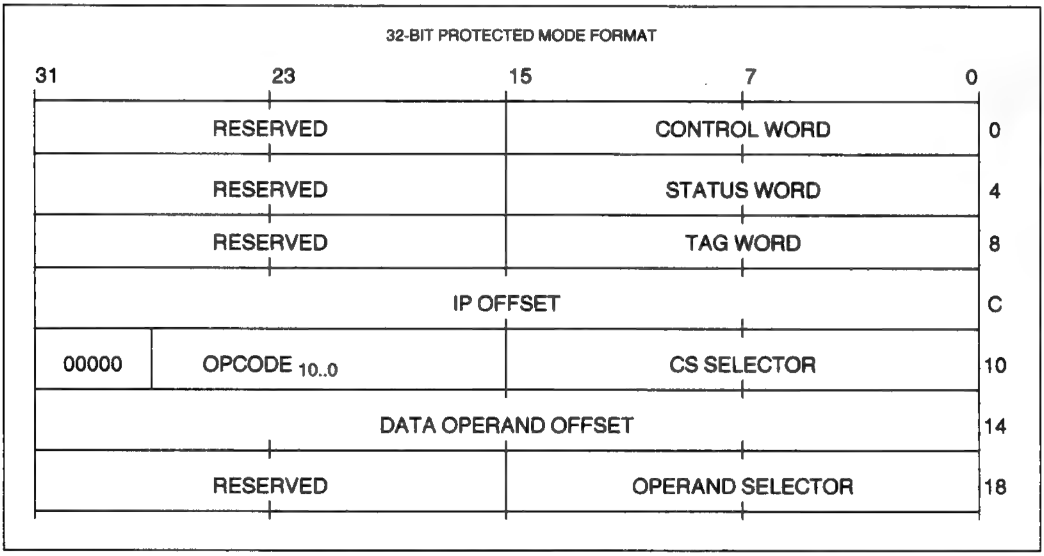


Figure 3-7. Instruction and Data Pointer Image in Memory, 32-Bit Protected-Mode Format

16-BIT PROTECTED MODE FORMAT

15 7 0	
CONTROL WORD	0
STATUS WORD	2
TAG WORD	4
IP OFFSET	6
CS SELECTOR	8
OPERAND OFFSET	A
OPERAND SELECTOR	C

Figure 3-8. Instruction and Data Pointer Image in Memory, 16-Bit Protected-Mode Format

32-BIT REAL-ADDRESS MODE FORMAT

31	23	15	7 0	
RESERVED		CONTROL WORD		0
RESERVED		STATUS WORD		4
RESERVED		TAG WORD		8
RESERVED		INSTRUCTION POINTER 15..0		C
0000	INSTRUCTION POINTER 31..16	0	OPCODE 10..0	10
RESERVED		OPERAND POINTER 15..0		14
0000	OPERAND POINTER 31..16	0000	00000000	18

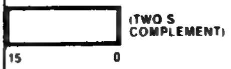
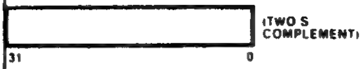
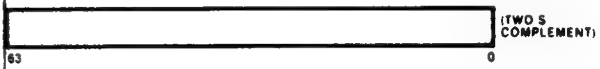
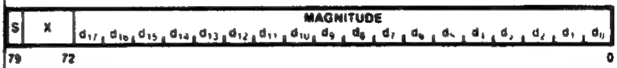
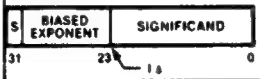
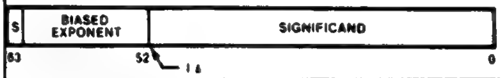
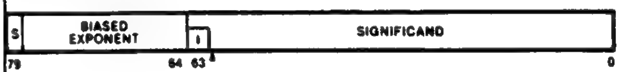
Figure 3-9. Instruction and Data Pointer Image in Memory, 32-Bit Real-Mode Format

16-BIT REAL-ADDRESS MODE AND VIRTUAL 8086 MODE FORMAT

15		7 0	
CONTROL WORD			0
STATUS WORD			2
TAG WORD			4
INSTRUCTION POINTER 15..0			6
IP19.16	0	OPCODE 10..0	8
OPERAND POINTER 15..0			A
DP 19.16	0	0 0 0 0 0 0 0 0 0 0 0	C

Figure 3-10. Instruction and Data Pointer Image in Memory, 16-Bit Real-Mode Format

Table 3-6. Intel387 SL Mobile Math CoProcessor Data Type Representation in Memory

Data Formats	Range	Precision	Most Significant Byte = HIGHEST ADDRESSED BYTE 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0
Word Integer	$\pm 10^4$	16 Bits	
Short Integer	$\pm 10^9$	32 Bits	
Long Integer	$\pm 10^{18}$	64 Bits	
Packed BCD	$\pm 10^{18}$	18 Digits	 S X MAGNITUDE d17 d16 d15 d14 d13 d12 d11 d10 d9 d8 d7 d6 d5 d4 d3 d2 d1 d0 79 72 0
Single Precision	$\pm 10^{\pm 38}$	24 Bits	
Double Precision	$\pm 10^{\pm 308}$	53 Bits	
Extended Precision	$\pm 10^{\pm 4932}$	64 Bits	 S BIASED EXPONENT I SIGNIFICAND 79 64 63 0

290427-10

NOTES:

1. S = Sign bit (0 = positive, 1 = negative)
2. d_n = Decimal digit (two per byte)
3. X = Bits have no significance; Math CoProcessor ignores when loading, zeros when storing
4. ▲ = Position of implicit binary point
5. I = Integer bit of significand; stored in temporary real, implicit in single and double precision
6. Exponent Bias (normalized values):
   Single: 127 (7FH)
   Double: 1023 (3FFH)
   Extended REal: 16383 (3FFFH)
7. Packed BCD: $(-1)^S$ $(D_{17}..D_0)$
8. Real: $(-1)^S$ $(2^{E\text{-}BIAS})$ $(F_0\ F_1...)$

3.3 Data Types

Table 3-6 lists the seven data types that the Math CoProcessor supports and presents the format for each type. Operands are stored in memory with the least significant digit at the lowest memory address. Programs retrieve these values by generating the lowest address. For maximum system performance, all operands should start at physical-memory addresses that correspond to the word size of the CPU; operands may begin at any other addresses, but will require extra memory cycles to access the entire operand.

The data type formats can be divided into three classes: binary integer, decimal integer, and binary real. These formats, however, exist in memory only. Internally, the Math CoProcessor holds all numbers in the extended-precision real format. Instructions that load operands from memory automatically convert operands represented in memory as 16, 32, or 64-bit integers, 32 or 64-bit floating point numbers, or 18 digit packed BCD numbers into extended-precision real format. Instructions that store operands in memory perform the inverse type conversion.

In addition to the typical real and integer data values, the Intel387 SL Mobile Math CoProcessor data formats encompass encodings for a variety of special values. These special values have significance and can express relevant information about the computations or operations that produced them. The various types of special values are denormal real numbers, zeros, positive and negative infinity, NaNs (Not-a-Number), Indefinite, and unsupported formats. For further information on data types and formats, see the Intel387 Programmer's Reference Manual.

3.4 Interrupt Description

CPU interrupts are used to report errors or exceptional conditions while executing numeric programs in either real or protected mode. Table 3-7 shows these interrupts and their functions.

Table 3-7. CPU Interrupt Vectors Reserved for Math CoProcessor

Interrupt Number	Cause of Interrupt
7	An ESC instruction was encountered when EM or TS of CPU control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either an ESC or WAIT instruction causes interrupt 7. This indicates that the current Math CoProcessor context may not belong to the current task.
9	In a protected-mode system, an operand of a coprocessor instruction wrapped around an addressing limit (0FFFFH for expand-up segments, zero for expand-down segments) and spanned inaccessible addresses(1). The failing numerics instruction is not restartable. The address of the failing numerics instruction and data operand may be lost; an FSTENV does not return reliable addresses. The segment overrun exception should be handled by executing an FNINIT instruction (i.e., an FINIT without a preceding WAIT). The exception can be avoided by never allowing numerics operands to cross the end of a segment.
13	In a protected-mode system, the first word of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the ESC instruction that caused the exception, including any prefixes. The Math CoProcessor has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only ESC and WAIT instructions can cause this interrupt. The CPU return address pushed onto the stack of the exception handler points to a WAIT or ESC instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the Math CoProcessor. FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE cannot cause this interrupt.

NOTE:

1. An operand may wrap around an addressing limit when the segment limit is near an addressing limit and the operand is near the largest valid address in the segment. Because of the wrap-around, the beginning and ending addresses of such an operand will be at opposite ends of the segment. There are two ways that such an operand may also span inaccessible addresses: 1) if the segment limit is not equal to the addressing limit (e.g. addressing limit is FFFFH and segment limit is FFFDH) the operand will span addresses that are not within the segment (e.g. an 8-byte operand that starts at valid offset FFFCH will span addresses FFFC–FFFFH and 0000-0003H; however addresses FFFEH and FFFFH are not valid, because they exceed the limit); 2) if the operand begins and ends in present and accessible segments but intermediate bytes of the operand fall in a not-present page or in a segment or page to which the procedure does not have access rights.

3.5 Exception Handling

The Math CoProcessor detects six different exception conditions that occur during instruction execution. Table 3-8 lists the exception conditions in order of precedence, showing for each the cause and the default action taken by the Math CoProcessor if the exception is masked by its corresponding mask bit in the control word.

Any exception that is not masked by the control word sets the corresponding exception flag of the status word, sets the ES bit of the status word, and asserts the ERROR# signal. When the CPU attempts to execute another ESC instruction or WAIT, exception 16 occurs. The exception condition must be resolved via an interrupt service routine. The return address pushed onto the CPU stack upon entry to the service routine does not necessarily point to the failing instruction nor to the following instruction. The CPU saves the address of the floating-point instruction that caused the exception and the address of any memory operand required by that instruction.

Table 3-8. Intel387 SL Mobile Math CoProcessor Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signalling NaN, unsupported format, indeterminate for (0-∞, 0/0, (+∞) + (−∞), etc.), or stack overflow/underflow (SF is also set).	Result is a quiet NaN, integer indefinite, or BCD indefinte
Denormalized Operand	At least one of the operands is denormalized, i.e., it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number.	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format.	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes the loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g. 1/3); the result is rounded according to the rounding mode.	Normal processing continues

3.6 Initialization

After FNINIT or RESET, the control word contains the value 037FH (all exceptions masked, precision control 64 bits, rounding to nearest) the same values as in an Intel287 after RESET. For compatibility with the 8087 and Intel287, the bit that used to indicate infinity control (bit 12) is set to zero; however, regardless of its setting, infinity is treated in the affine sense. After FNINIT or RESET, the status word is initialized as follows:

- All exceptions are set to zero.
- Stack TOP is zero, so that after the first push the stack top will be register seven (111B).
- The condition code C_3-C_0 is undefined.
- The B-bit is zero.

The tag word contains FFFFH (all stack locations are empty).

The Intel386 Microprocessor and Intel387 Math CoProcessor initialization software must execute a FNINIT instruction (i.e., FINIT without a preceding WAIT) after RESET. The FNINIT is not strictly required for the Intel386 software, but Intel recommends its use to help ensure upware compatibility with other processors. After a hardware RESET, the ERROR# output is asserted to indicate that an Intel387 Math CoProcessor is present. To accomplish this, the IE (Invalid Exception) and ES (Error Summary) bits of the status word are set, and the IM bit (Invalid Exception Mask) in the control word is cleared. After FNINIT, the status word and the control word have the same values as in an Intel287 Math CoProcessor after RESET.

Immediately after FINIT, the FSTDW AX and FSTSG AX instructions may be executed if desired. FSTDW AX can supply information useful to programming system power management controllers. For example, after determining that the Math CoProcessor is fully static (Device Word register, bit 8), the Intel386 SL Microprocessor's Idle Math CoProcessor clock field should be programmed for the stop clock option to obtain maximum power savings.

3.7 Processing Modes

The Intel387 SL Mobile Math CoProcessor works the same whether the CPU is executing in real-addressing mode, protected mode, or virtual-8086 mode. All references to memory for numerics data or status information are performed by the CPU, and therefore obey the memory-management and protection rules of the CPU mode currently in effect. The Intel387 SL Mobile Math CoProcessor merely operates on instructions and values passed to it by the CPU and therefore is not sensitive to the processing mode of the CPU.

The real-address mode and virtual-8086 mode, the Intel387 SL Mobile Math CoProcessor is completely upward compatible with software for the 8086/8087 and 80286/80287 real-address mode systems.

In protected mode, the Intel387 SL Mobile Math CoProcessor is completely upward compatible with software for the 80286/80287 protected mode system.

The only differences of operation that may appear when 8086/8087 programs are ported to the protected mode (not using virtual-8086 mode) is in the format of operands for the administrative instructions FLDENV, FSTENV, FRSTOR, and FSAVE.

3.8 Programming Support

To use the Intel387 SL Mobile Math CoProcessor requires no special programming tools, because all new instructions and data types are directly supported by the assembler and compilers for high-level languages (except FSTDW AX and FSTSG AX; these can be generated with the ASM construct DW, define word). All Intel386 Microprocessor development tools that support Intel387 Math CoProcessor programs can also be used to develop software for the Intel386 SX and Intel386 SL Microprocessors and Intel387 SL Mobile Math CoProcessors. All 8086/8088 development tools that support the 8087 can also be used to develop software for the CPU and Math CoProcessor in real-address mode or virtual-8086 mode. All 80286 development tools that support the Intel287 Math CoProcessor can also be used to develop software for the Intel386 CPU and Intel387 Math CoProcessor.

The Intel387 SL Mobile Math CoProcessor supports all Intel387 Math CoProcessor instructions. The Intel386 SX CPU or Intel386 SL CPU and the Intel387 SL Mobile Math CoProcessor supports all the same programs and gives the same results as an Intel386 Microprocessor and Intel387 Math CoProcessor.

4.0 HARDWARE SYSTEM INTERFACE

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

4.1 Signal Description

In the following signal descriptions, the Intel387 SL Mobile Math CoProcessor pins are grouped by function as shown by Table 4-1. Table 4-1 lists every pin by its identifier, gives a brief description and lists some of its characteristics (Refer to Figure 1-1 and Table 1-1 for pin configuration).

All output signals can be tri-stated by driving STEN inactive. The output buffers of the bi-directional data pins D15–D0 are also tri-state; they only leave the floating state during read cycles when the Math CoProcessor is selected.

4.1.1 Intel386 CPU CLOCK 2 (CPUCLK2)

This input uses the CLK2 signal of the CPU to time the bus control logic. Several other Math CoProcessor signals are referenced to the rising edge of this signal. When CKM = 1 (synchronous mode) this pin also clocks the data interface and control unit and the floating point unit of the Math CoProcessor. This pin requires CMOS-level input. The signal on this pin is divided by two to produce the internal clock signal CLK.

4.1.2 Intel387 MATH COPROCESSOR CLOCK 2 (NUMCLK2)

When CKM = 0 (asynchronous mode), this pin provides the clock for the data interface and control unit and the floating point unit of the Math CoProcessor. In this case, the ratio of the frequency of NUMCLK2 to the frequency of CPUCLK2 must lie within the range 10:16 to 14:10 and the maximum frequency must not exceed the device specifications. When CKM = 1 (synchronous mode), signals on this pin are ignored: CPUCLK2 is used instead for the data interface and control unit and the floating point unit. This pin requires CMOS level input and should be tied low if not used.

Table 4-1. Pin Summary

Pin Name	Function	Active State	Input/ Output	Referenced To . . .
Execution Control				
CPUCLK2	Microprocessor Clock2		I	
NUMCLK2	Math CoProcessor Clock2		I	
CKM	Math CoProcessor Clock Mode		I	
RESETIN	System Reset	High	I	CPUCLK2
Math CoProcessor Handshake				
PEREQ	Processor Request	High	O	CPUCLK2
BUSY#	Busy Status	Low	O	CPUCLK2
ERROR#	Error Status	Low	O	NUMCLK2
Bus Interface				
D15–D0	Data Pins		I/O	CPUCLK2
W/R#	Write/Read Bus Cycle	High/Low	I	CPUCLK2
ADS#	Address Strobe	Low	I	CPUCLK2
READY#	Bus Ready Input	Low	I	CPUCLK2
READYO#	Ready Output	Low	O	CPUCLK2
Chip/Port Select				
STEN	Status Enable	High	I	CPUCLK2
NPS1#	Numerics Select #1	Low	I	CPUCLK2
NPS2	Numerics Select #2	High	I	CPUCLK2
CMD0#	Command	Low	I	CPUCLK2
Power and Ground				
V_{CC}	System Power			
V_{SS}	System Ground			

4.1.3 CLOCKING MODE (CKM)

This pin is strapping option. When it is strapped to V_{CC} (HIGH), the Math CoProcessor operates in synchronous mode; when strapped to V_{SS} (LOW), the Math CoProcessor operates in asynchronous mode. These modes relate to clocking of the internal data interface and control unit and the floating point unit only; the bus control logic always operates synchronously with respect to the CPU.

Synchronous mode requires the use of only one clock, the CPU's CLK2. Use of synchronous mode eliminates one clock generator from the board design and is recommended for all designs. Synchronous mode also allows the internal Power Management Unit to enable the idle and standby power saving modes.

Asynchronous mode is provided to maintain compatibility with the Intel387 SX Math CoProcessor. Asynchronous mode can provide higher performance of the floating point unit by running a faster clock on NUMCLK2. (The CPU's CLK2 must still be connected to CPUCLK2 input.) This which allows the floating point unit to run up to 40% faster than in synchronous mode. Internal power management is disabled in asynchronous mode and the Math CoProcessor will not support slow or stop clock power saving modes.

4.1.4 SYSTEM RESET (RESETIN)

A LOW to HIGH transition on this pin causes the Math CoProcessor to terminate its present activity and to enter a dormant state. RESETIN must remain active (HIGH) for at least 40 CPUCLK2 (NUMCLK2 if CKM = 0) periods.

The HIGH to LOW transitions of RESETIN must be synchronous with CPUCLK2, so that the phase of the internal clock of the bus control logic (which is the CPUCLK2 divided by two) is the same as the phase of the internal clock of the CPU. After RESETIN goes LOW, at least 50 CPUCLK2 (NUMCLK2 if CKM = 0) periods must pass before the first Math CoProcessor instruction is written into the Math CoProcessor. This pin should be connected to the CPU RESET pin. Table 4-2 shows the status of the output pins during the reset sequence. After a reset, all output pins return to their inactive state except for ERROR# which remains active (for CPU recognition) until cleared.

Table 4-2. Output Pin Status during Reset

Pin Value	Pin Name
HIGH	READYO#, BUSY#
LOW	PEREQ, ERROR#
Tri-State OFF	D15–D0

4.1.5 PROCESSOR REQUEST (PEREQ)

When active, this pin signals to the CPU that the Math CoProcessor is ready for data transfer to/from its data FIFO. When all data is written to or read from the data FIFO, PEREQ is deactivated. This signal always goes inactive before BUSY# goes inactive. This signal is reference to CPUCLK2. It should be connected to the CPU PEREQ input pin.

4.1.6 BUSY STATUS (BUSY#)

When active, this pin signals to the CPU that the Math CoProcessor is currently executing an instruction. This signal is referenced to CPUCLK2. It should be connected to the CPU BUSY# input pin.

4.1.7 ERROR STATUS (ERROR#)

This pin reflects the ES bit of the status register. When active, it indicates that an unmasked exception has occurred. This signal can be changed to the inactive state only by the following instructions (without a preceding WAIT): FNINIT, FNCLEX, FNSTENV, FNSAVE, FLDCW, FLDENV, and FRSTOR. ERROR# is driven active during RESET to indicate to the CPU that the Math CoProcessor is present. This pin is referenced to NUMCLK2 (or CPUCLK2 if CKM = 1). It should be connected to the ERROR# pin of the CPU.

4.1.8 DATA PINS (D15–D0)

These bi-directional pins are used to transfer data and opcodes between the CPU and Math CoProcessor. They are normally connected directly to the corresponding CPU data pins. HIGH state indicates a value of one. D0 is the least significant data bit. Timings are referenced to rising edge of CPUCLK2.

4.1.9 WRITE/READ BUS CYCLE (W/R#)

This signal indicates to the Math CoProcessor whether the CPU bus cycle in progress is a read or a write cycle. This pin should be connected directly to the CPU's W/R# pin. HIGH indicates a write cycle to the Math CoProcessor; LOW a read cycle from the Math CoProcessor. This input is ignored if any of the signals STEN, NPS1#, or NPS2 are inactive. Setup and hold times are referenced to CPUCLK2.

4.1.10 ADDRESS STROBE (ADS#)

This input, in conjunction with the READY# input, indicates when the Math CoProcessor bus control logic may sample W/R# and the chip select signals. Setup and hold times are referenced to CPUCLK2. This pin should be connected to the ADS# pin of the CPU.

4.1.11 BUS READY INPUT (READY#)

This input indicates to the Math CoProcessor when a CPU bus cycle is to be terminated. It is used by the bus control logic to trace bus activities. Bus cycles can be extended indefinitely until terminated by READY#. This input should be connected to the same signal that drives the CPU's READY# input. Setup and hold times are referenced to CPUCLK2.

4.1.12 READY OUTPUT (READYO#)

This pin is activated at such a time that write cycles are terminated after two clocks (except FLDENV and FRSTOR) and read cycles after three clocks. In configurations where no extra wait states are required, this pin must directly or indirectly drive the READY# input of the CPU. Refer to the section entitled "BUS OPERATION" for details. This pin is activated only during bus cycles that select the Math CoProcessor. This signal is referenced to CPUCLK2.

(FLDENV and FRSTOR require data transfers larger than the FIFO. Therefore, PEREQ is activated for the duration of transferring 2 words of 32 bits and then deactivated until the FIFO is ready to accept two additional words. The length of the write cycles of the last operand word in each transfer as well as the first operand word transfer of the entire instruction is 3 clocks instead of 2 clocks. This is done to give the Intel386 CPU enough time to sample PEREQ and to notice that the Intel387 is **not** ready for additional transfers.)

4.1.13 STATUS ENABLE (STEN)

This pin serves as a chip select for the Math CoProcessor. When inactive, this pin forces BUSY#, PEREQ, ERROR# and READYO# outputs into a floating state. D15–D0 are normally floating and will leave the floating state only if STEN is active and additional conditions are met (read cycle). STEN also causes the chip to recognize its other chip select inputs. STEN makes it easier to do on-board testing (using the overdrive method) of other chips in systems containing the Math CoProcessor. STEN should be pulled up with a resistor so that it can be pulled down when testing. In boards that do not use on-board testing STEN should be connected to V_{CC}. Setup and hold times are relative to CPUCLK2. Note that STEN must maintain the same setup and hold times as NPS1#, NPS2, and CMD0# (i.e., if STEN changes state during a Math CoProcessor bus cycle, it must change state during the same CLK period as the NPS1#, NPS2, and CMD0# signals).

4.1.14 MATH COPROCESSOR SELECT 1 (NPS1#)

When active (along with STEN and NPS2) in the first period of a CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the Math CoProcessor. This pin should be connected directly to the M/IO# pin of the CPU, so that the Math CoProcessor is selected only when the CPU performs I/O cycles. Setup and hold times are referenced to the rising edge of CPUCLK2.

4.1.15 MATH COPROCESSOR SELECT 2 (NPS2)

When active (along with STEN and NPS1#) in the first period of a CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the Math CoProcessor. This pin should be connected directly to the A23 pin of the CPU, so that the Math CoProcessor is selected only when the CPU issues one of the I/O addresses reserved for the Math CoProcessor (8000F8h, 8000FCh, or 8000FEh which is treated as 8000FCh by the Math CoProcessor). Setup and hold times are referenced to the rising edge of CPUCLK2.

4.1.16 COMMAND (CMD0#)

During a write cycle, this signal indicates whether an opcode (CMD0# active low) or data (CMD0# inactive high) is being sent to the Math CoProcessor. During a read cycle, it indicates whether the control or status register (CMD0# active) or a data register (CMD0#) is being read. CMD0# should be connected directly to the A2 output of the CPU. Setup and hold times are referenced to the rising edge of CPUCLK2 at the end of PH2.

4.1.17 SYSTEM POWER (V_{CC})

System power provides the +5V DC supply input. All V_{CC} pins should be tied together on the circuit board and local decoupling capacitors should be used between V_{CC} and V_{SS}.

4.1.18 SYSTEM GROUND (V_{SS})

System ground provides the 0V connection from which all inputs and outputs are measured. All V_{SS} pins should be tied together on the circuit board and local decoupling capacitors should be used between V_{CC} and V_{SS}.

4.2 System Configuration

The Intel387 SL Mobile Math CoProcessor is designed to interface with the Intel386 SL Microprocessor SuperSet as shown in Figure 4-1 or the Intel386 SX Microprocessor. The Intel386 SL CPU math coprocessor interface has been optimized to provide a fully compatible PC design without additional hardware through dedicated control and clock signals.

The Intel386 SL CPU detects the presence of the math coprocessor during reset by sampling the ERROR# input. If the math coprocessor is not installed then the Intel386 SL CPU stops the Math CoProcessor clock. If the math coprocessor is detected, the CPU provides the clock output derived from its EFI clock input. ERROR# is also routed to the 82360 SL I/O component for interrupt generation to maintain PC compatibility.

The Intel386 SL CPU has dedicated control signals for the math coprocessor and these should not be used for any other device. READYO# of the Intel387 SL Mobile Math CoProcessor should route directly to the Intel386 SL CPU READY# input and the Intel387 SL Mobile Math CoProcessor READY# input. Logical ANDing of other system ready signals should not be done and a wait-state generator is not needed and should not be used.

The only signals that are shared in the Intel386 SL Microprocessor design are A2 and the data bus. The Intel386 SL CPU drives the cache memory with these signals in addition to the Math CoProcessor.

4.2.1 INTEL386 SX MICROPROCESSOR SYSTEM

The Intel386 SL Math CoProcessor is socket compatible with the Intel387 SX Math CoProcessor. No change to an existing design is necessary to use the Intel387 SL Mobile Math CoProcessor. Internal Power Management is automatically executed without hardware or software intervention.

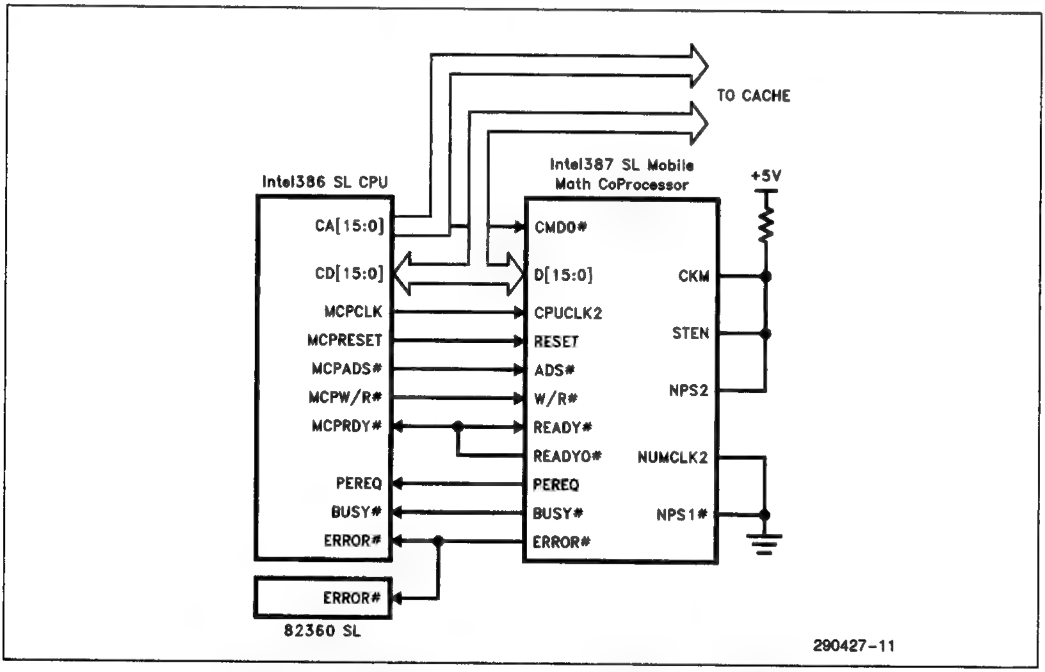


Figure 4-1. Intel386 SL CPU Math CoProcessor Interface

4.3 Math CoProcessor Architecture

As shown in Figure 2-1 Block Diagram, the Intel387 SL Mobile Math CoProcessor is internally divided into four sections; the Bus Control Logic (BCL), the Data Interface and Control Logic, the Floating Point Unit (FPU), and the Power Management Unit (PMU). The Bus Control Logic is responsible for the CPU bus tracking and interface. The BCL is the only unit in the Math CoProcessor that must run synchronously with the CPU; the rest of the Math CoProcessor can run asynchronously with respect to the CPU. The Data Interface and Control Unit is responsible for the data flow to and from the FPU and the control registers, for receiving the instructions, decoding them, sequencing the microinstructions, and for handling some of the administrative instructions. The Floating Point Unit (with the support of the control unit which contains the sequencer and other support units) executes the mathematical instructions. The Power Manager is new to the Intel387 family and is the nucleus of the static architecture. It is responsible for shutting down idle sections of the device to save power.

4.3.1 BUS CONTROL LOGIC

The BCL communicates solely with the CPU using I/O bus cycles. The BCL appears to the CPU as a special peripheral device. It is special in two respects: the CPU initiates I/O automatically when it encounters ESC instructions, and the CPU uses reserved I/O addresses to communicate with the BCL. The BCL does not communicate directly with memory. The CPU performs all memory access, transferring input operands from the memory to the Math CoProcessor and transferring outputs from the Math CoProcessor to memory.

4.3.2 DATA INTERFACE AND CONTROL UNIT

The data interface and control unit latches the data and, subject to BCL control, directs the data to the FIFO or the instruction decoder. The instruction decoder decodes the ESC instructions sent to it by the CPU and generates controls that direct the data flow in the FIFO. It also triggers the microinstruction sequencer that controls execution of each instruction. If the ESC instruction is FINIT, FCLEX, FSTSW, FSTSW AX, FSTCW, FSTDW AX, FSTSG AX, FSETPM, or FRSTPM, the control unit executes it independently of the FPU and the sequencer. The data interface and control unit is the unit that generates the BUSY#, PEREQ, and ERROR# signals that synchronize the Math CoProcessor activities with the CPU.

4.3.3 FLOATING POINT UNIT

The FPU executes all instructions that involve the register stack, including arithmetic, logical, transcendental, constant, and data transfer instructions. The data path in the FPU is 84 bits wide (68 significant bits, 15 exponent bits, and a sign bit) which allows internal operand transfers to be performed at very high speeds.

4.3.4 POWER MANAGEMENT UNIT

The Power Management Unit (PMU) controls all internal power savings circuits. When the Math CoProcessor is not executing an instruction, the PMU disables the internal clock to the FPU, Control Unit, and Data Interface within three clocks. The Bus Control Logic remains enabled to accept the next instruction. Upon decode of a valid Math CoProcessor bus cycle, the PMU enables the internal clock to all circuits. No loss in performance occurs.

4.4 Bus Cycles

All bus cycles are initiated by the CPU. The pins STEN, NPS1#, NPS2, CMD0, and W/R# identify bus cycles for the Math CoProcessor. Table 4-3 defines the types of Math CoProcessor bus cycles.

Table 4-3. Bus Cycle Definition

STEN	NPS1#	NPS2	CMD0#	W/R#	Bus Cycle Type
0	X	X	X	X	Math CoProcessor not selected and all outputs in floating state
1	1	X	X	X	Math CoProcessor not selected
1	X	0	X	X	Math CoProcessor not selected
1	0	1	0	0	CW or SW read from Math CoProcessor
1	0	1	0	1	Opcode write to Math CoProcessor
1	0	1	1	0	Data read from Math CoProcessor
1	0	1	1	1	Data write to Math CoProcessor

4.4.1 INTEL387 SL MATH COPROCESSOR ADDRESSING

The NPS1#, NPS2, and CMD0 signals allow the Math CoProcessor to identify which bus cycles are intended for the Math CoProcessor. The Math CoProcessor responds to I/O cycles when the I/O address is 8000F8h, 8000FCh, and 8000FEh (treated as 8000FCh). The Math CoProcessor responds to I/O cycles when bit 23 of the I/O address is set. In other words, the Math CoProcessor acts as an I/O device in a reserved I/O address space.

Because A23 is used to select the Intel387 SL Mobile Math CoProcessor for data transfers, it is not possible for a program running on the CPU to address the Math CoProcessor with an I/O instruction. Only ESC instructions cause the CPU to communicate with the Math CoProcessor.

4.4.2 CPU/MATH COPROCESSOR SYNCHRONIZATION

The pins BUSY#, PEREQ, and ERROR# are used for various aspects of synchronization between the CPU and the Math CoProcessor.

BUSY# is used to synchronize instruction transfer from the CPU to the Math CoProcessor. When the Math CoProcessor recognizes an ESC instruction it asserts BUSY#. For most ESC instructions, the CPU waits for the Math CoProcessor to deassert BUSY# before sending the new opcode.

The Math CoProcessor uses the PEREQ pin of the CPU to signal that the Math CoProcessor is ready for data transfer to or from its data FIFO. The Math CoProcessor does not directly access memory; rather, the CPU provides memory access services for the Math CoProcessor. (For this reason, memory access on behalf of the Math CoProcessor always obeys the protection rules applicable to the current CPU mode.) Once the CPU initiates an Math CoProcessor instruction that has operands, the CPU waits for PEREQ signals that indicate when the Math CoProcessor is ready for operand transfer. Once all operands have been transferred (or if the instruction has no operands) the CPU continues program execution while the Math CoProcessor executes the ESC instruction.

In 8087/8087 systems, WAIT instructions may be required to achieve synchronization of both commands and operands. In the Intel386 Microprocessor and Intel387 Math CoProcessor systems, however, WAIT instructions are required only for operand synchronization; namely, after Math CoProcessor stores to memory (except FSTSW and FSTCW) or load from memory. (In 80286/80287 systems, WAIT is required before FLDENV and FRSTOR.) Used this way, WAIT ensures that the value has already been written or read by the Math CoProcessor before the CPU reads or changes the value.

Once it has started to execute a numerics instruction and has transferred and operands from the CPU, the Math CoProcessor can process the instruction in parallel with and independent of the host CPU. When the Math CoProcessor detects an exception, it asserts the ERROR# signal, which causes a CPU interrupt.

4.4.3 SYNCHRONOUS/ASYNCHRONOUS MODES

The internal logic of the Math CoProcessor can operate either directly from the CPU clock (synchronous mode) or from a separate clock (asynchronous mode). The two configurations are distinguished by the CKM pin. In either case, the bus control logic (BCL) of the Math CoProcessor is synchronized with the CPU clock. Use of asynchronous mode allows the BCL and the FPU section of the Math CoProcessor to run at different speeds. In this case, the ratio of the frequency of NUMCLK2 to the frequency of CPUCLK2 must lie within the range 10:16 to 14:10. Use of synchronous mode eliminates one clock generator from the board design. The internal Power Management Unit of the Intel387 SL Mobile Math CoProcessor is disabled in asynchronous mode.

4.4.4 AUTOMATIC BUS CYCLE TERMINATION

In configurations where no extra wait states are required, READYO# can drive the CPU's READY# input and the Math CoProcessors READY# input. If wait states are required, this pin should be connected to the logic that ORs all READY outputs from peripheral devices on the CPU bus. READYO# is asserted by the Math CoProcessor only during I/O cycles that select the Math CoProcessor. Refer to Section 5.0 Bus Operation for details.

5.0 BUS OPERATION

With respect to bus interface, the Intel387 SL Mobile Math CoProcessor is fully synchronous with the CPU. Both operate at the same rate because each generates its internal CLK signal by dividing CPUCLK2 by two. Furthermore, both internal CLK signals are in phase, because they are synchronized by the same RESETIN signal.

A bus cycle for the Math CoProcessor starts when the CPU activates ADS# and drives new values on the address and cycle definition lines (W/R#, M/IO#, etc.). The Math CoProcessor examines the address and cycle definition lines in the same CLK period during which ADS# is activated. This CLK period is considered the first CLK of the bus cycle.

During this first CLK period, the Math CoProcessor also examines the W/R# input signal to determine whether the cycle is a read or a write cycle and examines the CMD0# input to determine whether an opcode, operand, or control/status register transfer is to occur.

The Intel387 SL Mobile Math CoProcessor supports both pipelined (i.e., overlapped) and non-pipelined bus cycles. A non-pipelined cycle is one for which the CPU asserts ADS# when no other bus cycle is in progress. A pipelined bus cycle is one for which the CPU asserts ADS# and provides valid next address and control signals before the prior Math CoProcessor cycle terminates. The CPU may do this as early as the second CLK period after asserting ADS# for the prior cycle. Pipelining increases the availability of the bus by at least one CLK period. The Intel387 SL Mobile Math CoProcessor supports pipelined bus cycles in order to optimize address pipelining by the CPU for memory cycles.

Bus operation is described in terms of an abstract state machine. Figure 5-1 illustrates the states and state transitions for Math CoProcessor bus cycles:

- T_I is the idle state. This is the state of the bus logic after RESET, the state to which bus logic returns after every non-pipelined bus cycle, and the state to which bus logic returns after a series of pipelined cycles.
- T_{RS} is the READY# sensitive state. Different types of bus cycles may require a minimum of one or two successive T_{RS} states. The bus logic remains in T_{RS} state until READY# is sensed, at which point the bus cycle terminates. Any number of wait states may be implemented by delaying READY#, thereby causing additional successive T_{RS} states.
- T_P is the first state for every pipelined bus cycle. This state is not used by non-pipelined cycles.

Note that the bus logic tracks bus state regardless of the values on the chip/port select pins. The READYO# output of the Math CoProcessor indicates when a Math CoProcessor bus cycle may be terminated if no extra wait states are required. For all write cycles (except those for the instructions FLDENV and FRSTOR), READYO# is always asserted during the first T_{RS} state, regardless of the number of wait states. For all read cycles (and write cycles for FLDENV and FRSTOR), READY# is always asserted in the second T_{RS} state, regardless of the number of wait states. These rules apply to both pipelined and non-pipelined cycles. Systems designers may use READYO# in one of the following ways:

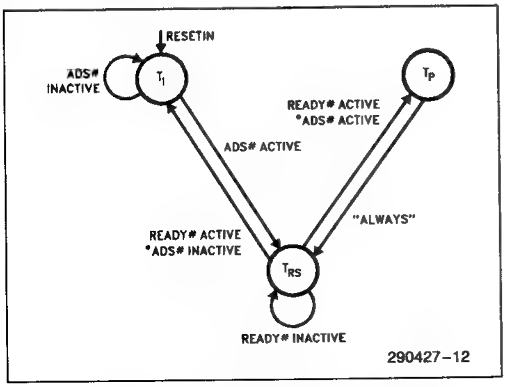


Figure 5-1. Bus State Diagram

1. Connect it (directly or through logic that ORs READY# signals from other devices) to the READY# inputs of the CPU and Math CoProcessor.
2. Use it as one input to a wait-state generator.

The following sections illustrate different types of Intel387 SL Mobile Math CoProcessor bus cycles. Because different instructions have different amounts of overhead before, between, and after operand transfer cycles, it is not possible to represent in a few diagrams all of the combinations of successive operand transfer cycles. The following bus cycle diagrams show memory cycles between Math CoProcessor operand transfer cycles. Note however that, during FRSTOR, some consecutive accesses to the Math CoProcessor do not have intervening memory accesses. For the timing relationship between operand transfer cycles and opcode write or other overhead activities, see Figure 7-7 "Other Parameters".

5.1 Non-Pipelined Bus Cycles

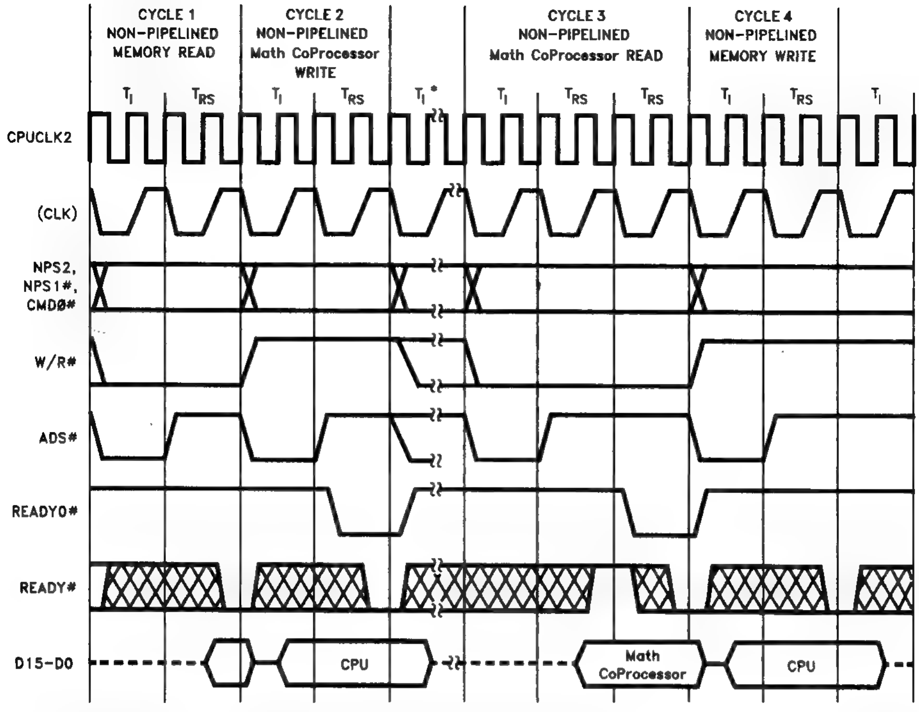
Figure 5-2 illustrates bus activity for consecutive non-pipelined bus cycles.

At the second clock of the bus cycle, the Math CoProcessor enters the T_{RS} state. During this state, it samples the READY# input and stays in this state as long as READY# is inactive.

5.1.1 WRITE CYCLE

In write cycles, the Math CoProcessor drives the READYO# signal for one CLK period during the second CLK period of the cycle (i.e., the first T_{RS} state); therefore, the fastest write cycle takes two CLK periods (see cycle 2 of Figure 5-2). For the instructions FLDENV and FRSTOR, however, the Math CoProcessor forces wait state by delaying the activation of READYO# to the second T_{RS} state (not shown in Figure 5-2).

The Math CoProcessor samples the D15–D0 inputs into data latches at the falling edge of CLK as long as it stays in T_{RS} state.



290427-13

Cycles 1 & 2 represent part of the operand transfer cycle for instructions involving either 4-byte or 8-byte operand loads.
Cycles 3 & 4 represent part of the operand transfer cycle for a store operation.
*Cycles 1 & 2 could repeat here or T_I states for various non-operand transfer cycles and overhead.

Figure 5-2. Nonpipelined Read and Write Cycles

When READY# is asserted, the Math CoProcessor returns to the idle state. Simultaneously with the Math CoProcessor entering the idle state, the CPU may assert ADS# again, signaling the beginning of yet another cycle.

5.1.2 READ CYCLE

At the rising edge of CLK in the second CLK period of the cycle (i.e., the first T_{RS} state), the Math CoProcessor starts to drive the D15–D0 outputs and continues to drive them as long as it stays in T_{RS} state.

At least one wait state must be inserted to ensure that the CPU latches the correct data. Because the Math CoProcessor starts driving the data bus only at the rising edge of CLK in the second clock period of the bus cycle, not enough time is left for the data signals to propagate and be latched by the CPU before the next falling edge of CLK. Therefore, the Math CoProcessor does not drive the READYO# signal until the third CLK period of the cycle. Thus, if the READYO# output drives the CPU's READY# input, one wait state is automatically inserted.

Because one wait state is required for Math CoProcessor reads, the minimum length of an Math CoProcessor read cycle is three CLK periods, as cycle 3 of Figure 5-2 shows.

When READY# is asserted, the Math CoProcessor returns to the idle state. Simultaneously with the Math CoProcessor's entering the idle state, the CPU may assert ADS# again, signaling the beginning of yet another cycle. The transition from T_{RS} state to idle state causes the Math CoProcessor to put the D15–D0 outputs into the floating state, allowing another device to drive the data bus.

5.2 Pipelined Bus Cycles

Because all the activities of the Math CoProcessor bus interface occur either during the T_{RS} state or during the transitions to or from that state, the only difference between a pipelined and a non-pipelined cycle is the manner of changing from one state to another. The exact activities during each state are detailed in the previous section "Non-pipelined Bus Cycles".

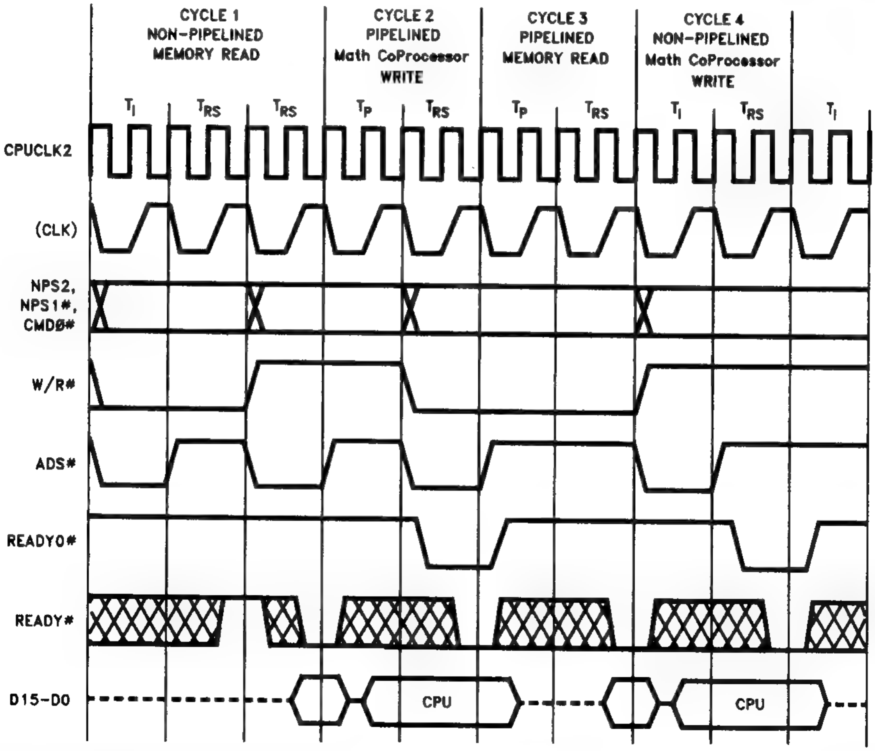
When the CPU asserts ADS# before the end of a bus cycle, both ADS# and READY# are active during a T_{RS} state. This condition causes the Math CoProcessor to change to a different state named T_P. One clock period.after a T_P state, the Math CoProcessor always returns to the T_{RS} state. In consecutive pipelined cycles, the Math CoProcessor bus logic uses only the T_{RS} and T_P states.

Figure 5-3 shows the fastest transitions into and out of the pipelined bus cycles. Cycle 1 in the figure represents a non-pipelined cycle. (Non-pipelined write are always followed by another non-pipelined cycle, because READY# is asserted before the earliest possible assertion of ADS# for the next cycle.)

Figure 5-4 shows pipelined write and read cycles with one additional T_{RS} state beyond the minimum required. To delay the assertion of READY# requires external logic.

5.3 Mixed Bus Cycles

When the Math CoProcessor bus logic is in the T_{RS} state, it distinguishes between non-pipelined and pipelined cycles according to the behavior of ADS# and READY#. In a non-pipelined cycle, only READY# is activated, and the transition is from the T_{RS} state to the idle state. In a pipelined cycle, both READY# and ADS# are active, and the transition is first from T_{RS} state to T_P state, then, after one clock period, back to T_{RS} state.

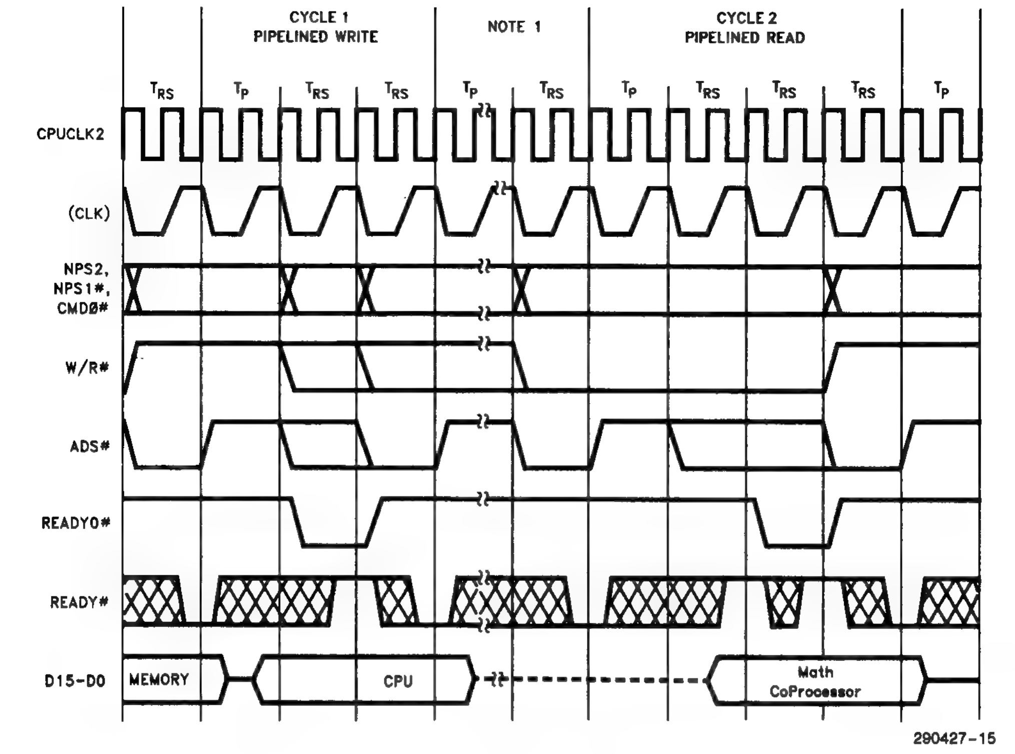


290427-14

Cycle 1–Cycle 4 represent the operand transfer cycle for an instruction involving a transfer of two 32-bit loads in total. The opcode write cycles and other overhead are not shown.

Note that the next cycle will be a pipelined cycle if both READY# and ADS# are sampled active at the end of a T_{RS} state of the current cycle.

Figure 5-3. Fastest Transitions to and from Pipelined Cycles

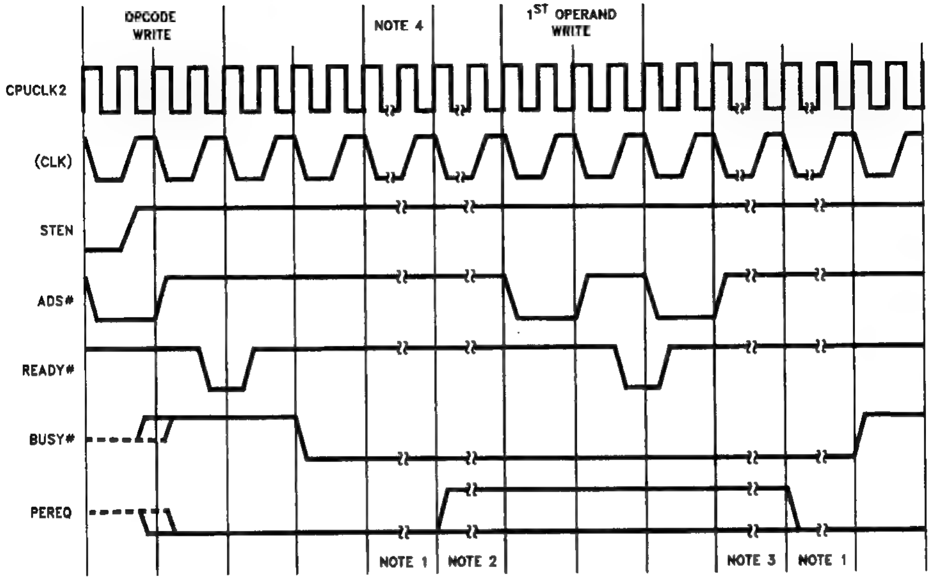


NOTE:
1. Cycles between operand write to the Math CoProcessor and storing result.

Figure 5-4. Pipelined Cycles with Wait States

5.4 BUSY# and PEREQ Timing Relationship

Figure 5-5 shows the activation of BUSY# at the beginning of instruction execution and its deactivation upon completion of the instruction. PEREQ is activated within this interval. If ERROR# is ever asserted, it would be asserted at least six CPUCLK2 periods after the deactivation of PEREQ and would be deasserted at least six CPUCLK2 periods before the deactivation of BUSY#.



290427-16

NOTES:
1. Instruction dependent.
2. PEREQ is an asynchronous input to the Intel386™ Microprocessor; it may not be asserted (instruction dependent).
3. More operand transfers.
4. Memory read (operand) cycle is not shown.

Figure 5-5. STEN, BUSY#, and PEREQ Timing Relationships

6.0 PACKAGE SPECIFICATIONS

6.1 Mechanical Specifications

The Intel387 SL Mobile Math CoProcessor is packaged in a 68-pin PLCC package. Detailed mechanical specifications can be found in the Intel Packaging Specification, Order Number 231369.

6.2 Thermal Specifications

The Intel387 SL Mobile Math CoProcessor is specified for operation when the case temperature is within the range of 0°C to 100°C. The case temperature (T_C) may be measured in any environment to determine whether the Intel387 SL Mobile Math CoProcessor is within the specified operating range. The case temperature should be measured at the center of the top surface.

The ambient temperature (T_A) is guaranteed as long as T_C is not violated. The ambient temperature can be calculated from the θ_{JC} (thermal resistance constant from the transistor junction to the case) and θ_{JA} (thermal resistance from junction to ambient) from the following calculations:

Junction Temperature $T_J = T_C + P^{*}\theta_{JC}$

Ambient Temperature $T_A = T_J - P^{*}\theta_{JA}$

Case Temperature $T_C = T_A + P^{*}(\theta_{JA} - \theta_{JC})$

Values for θ_{JA} and θ_{JC} are given in Table 6-1 for the 68 pin PLCC package. θ_{JC} is given at various airflows. Table 6-2 shows the maximum T_A allowable without exceeding T_C at various airflows. Note that T_A can be improved further by attaching a heat sink to the package. P is calculated by using the maximum hot I_{CC} and maximum V_{CC}.

Table 6-1. Thermal Resistances (°C/Watt) θ_{JC} and θ_{JA}

Package	θ_{JC}	θ_{JA} versus Airflow - ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
68-Pin PLCC	8	30	25	20	15.5	13	12

Table 6-2. Maximum T_A at Various Airflows

Package		T_A (°C) versus Airflow - ft/min (m/sec)				
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
68-Pin PLCC	84.9	88.3	91.8	94.8	96.6	97.2

Maximum T_A is calculated at maximum V_{CC} and maximum I_{CC}.

7.0 ELECTRICAL CHARACTERISTICS

The following specifications represent the targets of the design effort. They are subject to change without notice. Contact your Intel representative to get the most up-to-date values.

7.1 Absolute Maximum Ratings*

Case Temperature T_C Under Bias . . . 0°C to +100°C

Storage Temperature . . . . −65°C to +150°C

Voltage on Any Pin
with Respect to Ground . . . . −0.5 to V_{CC} + 0.5

Power Dissipation 0.8W

NOTICE: This data sheet contains information on products in the sampling and initial production phases of development. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest data sheet before finalizing a design.

******WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

7.2 D.C. Characteristics

Table 7-1. D.C. Specifications $T_C = 0°C$ to $+100°C$, $V_{CC} = 5V \pm 10\%$

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input LO Voltage	−0.3	+0.8	V	(Note 1)
V_{IH}	Input HI Voltage	2.0	$V_{CC}+0.3$	V	(Note 1)
V_{CL}	CPUCLK2 and NUMCLK2 Input LO Voltage	−0.3	+0.8	V	
V_{CH}	CPUCLK2 and NUMCLK2 Input HI Voltage	$V_{CC}-0.8$	$V_{CC}+0.8$	V	
V_{OL}	Output LO Voltage		0.45	V	(Note 2)
V_{OH}	Output HI Voltage	2.4		V	(Note 3)
V_{OH}	Output HI Voltage	$V_{CC}-0.8$		V	(Note 4)
I_{CC}	Power Supply Current				
	Dynamic Mode				
	Freq. = 25 MHz(5)		150	mA	ICC typ. = 130 mA
	Freq. = 20 MHz(5)		125	mA	ICC typ. = 110 mA
	Freq. = 16 MHz(5)		100	mA	ICC typ. = 90 mA
	Freq. = 1 MHz(5)		20	mA	ICC typ. = 5 mA
	Idle Mode(6)		7	mA	ICC typ. = 4 mA
	Standby Mode(7)		100	μA	ICC typ. = 25 μA
I_{LI}	Input Leakage Current		±15	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	I/O Leakage Current		±15	μA	$0.45V \leq V_O \leq V_{CC}$
C_{IN}	Input Capacitance	7	10	pF	$f_c = 1$ MHz
C_O	I/O Capacitance	7	12	pF	$f_c = 1$ MHz
C_{CLK}	Clock Capacitance	7	20	pF	$f_c = 1$ MHz

NOTES:

1. This parameter is for all inputs, excluding the clock inputs.
2. This parameter is measured at I_{OL} as follows:
   Data = 4.0 mA
   READYO#, ERROR#, BUSY#, PEREQ = 25 mA
3. This parameter is measured at I_{OH} as follows:
   Data = 1.0 mA
   READYO#, ERROR#, BUSY#, PEREQ = 0.6 mA
4. This parameter is measured at I_{OH} as follows:
   Data = 0.2 mA
   READYO#, ERROR#, BUSY# PEREQ = 0.12 mA
5. Synchronous Clock Mode (CKM = 1). I_{CC} is measured at steady state, maximum capacitive loading on the outputs, and worst-case D.C. level at the inputs.
6. Intel387 SL Mobile Math CoProcessor Internal Idle Mode. Synchronous clock mode, clock and control inputs are active but the Math CoProcessor is not executing an instruction. Outputs driving CMOS inputs.
7. Supports the Intel386 SL CPU Power Management Feature.
   Synchronous clock mode, clock is stopped, and all control inputs are inactive. CMOS loads on outputs.

7.3 A.C. Characteristics

Table 7-2a. Timing Requirements of the Bus Interface Unit

T_C = 0°C to +100°C, V_{CC} = 5V ±10% (All measurements made at 1.5V unless otherwise specified)

Pin	Symbol	Parameter	Min (ns)	Max (ns)	Test Conditions	Refer to Figure
CPUCLK2	t1	Period	20	DC	2.0V	7.2
CPUCLK2	t2a	High Time	6		2.0V	
CPUCLK2	t2b	High Time	3		$V_{CC}-0.8V$	
CPUCLK2	t3a	Low Time	6		2.0V	
CPUCLK2	t3b	Low Time	4		0.8V	
CPUCLK2	t4	Fall Time		7	From $V_{CC}-0.8V$ to 0.8V	
CPUCLK2	t5	Rise Time		7	From 0.8V to $V_{CC}-0.8V$	
READYO#	t7a	Out Delay	4	25	C_L = 50 pF	7.3
PEREQ	t7b	Out Delay	4	23	C_L = 50 pF	
BUSY#	t7c	Out Delay	4	23	C_L = 50 pF	
ERROR#	t7d	Out Delay	4	23	C_L = 50 pF	
D15–D0	t8	Out Delay	1	45	C_L = 50 pF	7.4
D15–D0	t10	Setup Time	11			
D15–D0	t11	Hold Time	11			
D15–D0	t12*	Float Time	5	24		
READYO#	t13a*	Float Time	1	40		7.6
PEREQ	t13b*	Float Time	1	40		
BUSY#	t13c*	Float Time	1	40		
ERROR#	t13d*	Float Time	1	40		
ADS#	t14a	Setup Time	15			7.4
ADS#	t15a	Hold Time	4			
W/R#	t14b	Setup Time	15			
W/R#	t15b	Hold Time	4			
READY#	t16a	Setup Time	10			7.4
READY#	t17a	Hold Time	4			
CMD0#	t16b	Setup Time	16			
CMD0#	t17b	Hold Time	2			
NPS1#, NPS2	t16c	Setup Time	16			
NPS1#, NPS2	t17c	Hold Time	2			
STEN	t16d	Setup Time	15			
STEN	t17d	Hold Time	2			
RESETIN	t18	Setup Time	8			7.5
RESETIN	t19	Hold Time	3			

NOTE:

*Float condition occurs when maximum output current becomes less than ILO in magnitude. Float delay is not tested.

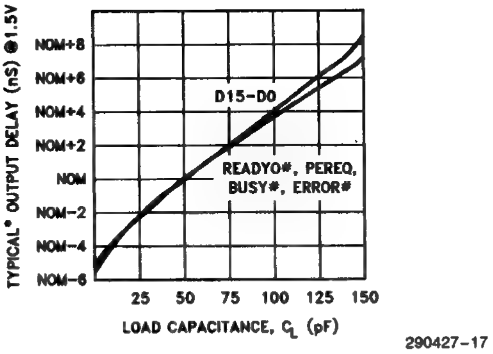
Table 7-2b. Timing Requirements of the Execution Unit (Asynchronous Mode CKM = 0)

Pin	Symbol	Parameter	Min	Max	Test Conditions	Refer to Figure
NUMCLK2	t1	Period	20	500	2.0V	7.2
NUMCLK2	t2a	High Time	6		2.0V	
NUMCLK2	t2b	High Time	3		$V_{CC}-0.8V$	
NUMCLK2	t3a	Low Time	6		2.0V	
NUMCLK2	t3b	Low Time	4		0.8V	
NUMCLK2	t4	Fall Time		7	From $V_{CC}-0.8V$ to 0.8V	
NUMCLK2	t5	Rise Time		7	From 0.8V to $V_{CC}-0.8V$	
NUMCLK2/ CPUCLK2		Ratio	10/16	14/10		

NOTE:
If not used (CKM = 1) tie NUMCLK2 low.

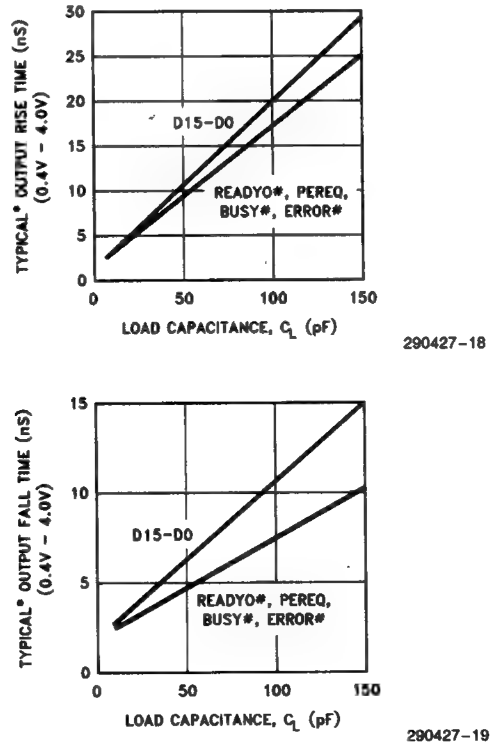
Table 7-2c. Other A.C. Parameters

Pin	Symbol	Parameter	Min	Max	Units
RESETIN	t30	Duration	40		NUMCLK2
RESETIN	t31	RESETIN Inactive to 1st Opcode Write	50		NUMCLK2
BUSY#	t32	Duration	6		CPUCLK2
BUSY#, ERROR#	t33	ERROR# (In)Active to BUSY# Inactive	6		CPUCLK2
PEREQ, ERROR#	t34	PEREQ Inactive to ERROR# Active	6		CPUCLK2
READY#, BUSY#	t35	READY# Active to BUSY# Active	0	4	CPUCLK2
READY#	t36	Minimum Time from Opcode Write to Opcode/Operand Write	4		CPUCLK2
READY#	t37	Minimum Time from Operand Write to Operand Write	4		CPUCLK2



NOTE:
*Typical part under worst-case conditions.

Figure 7-1a. Typical Output Valid Delay vs Load Capacitance at Max Operating Temperature



NOTE:
*Typical part under worst-case conditions.

Figure 7-1b. Typical Output Slew Time vs Load Capacitance at Max Operating Temperature

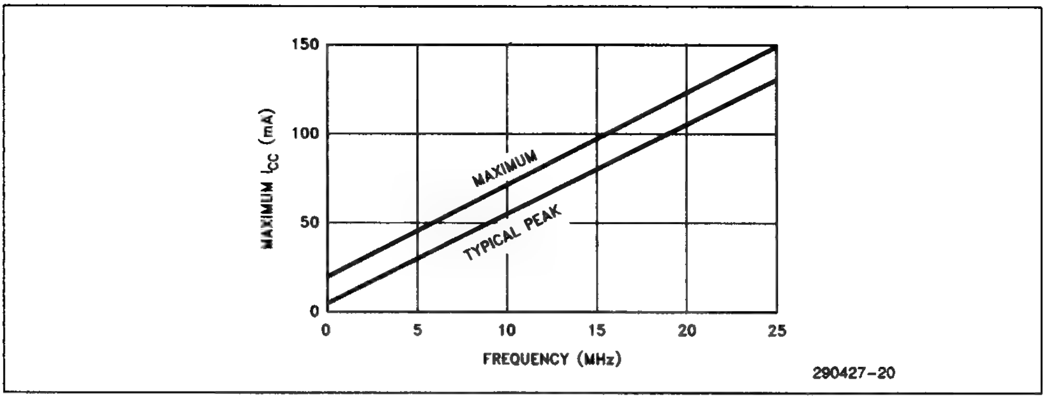


Figure 7-1c. Maximum I_{CC} vs Frequency

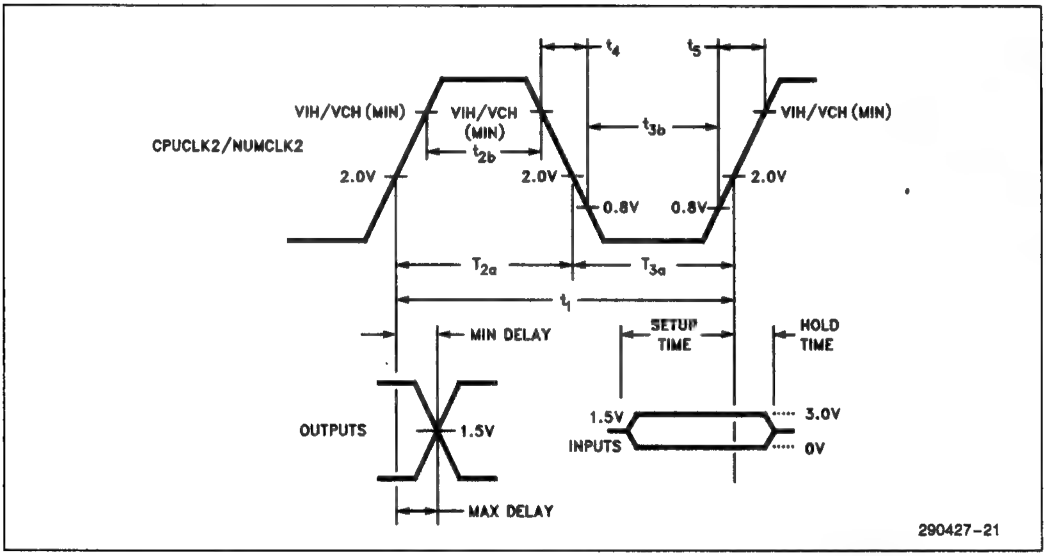


Figure 7-2. CPUCLK2/NUMCLK2 Waveform and Measurement Points for Input/Output

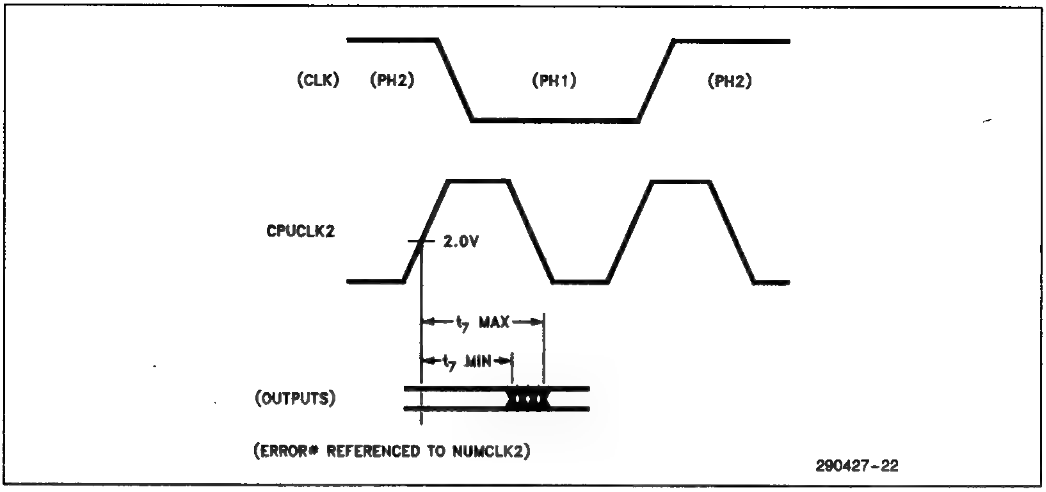


Figure 7-3. Output Signals

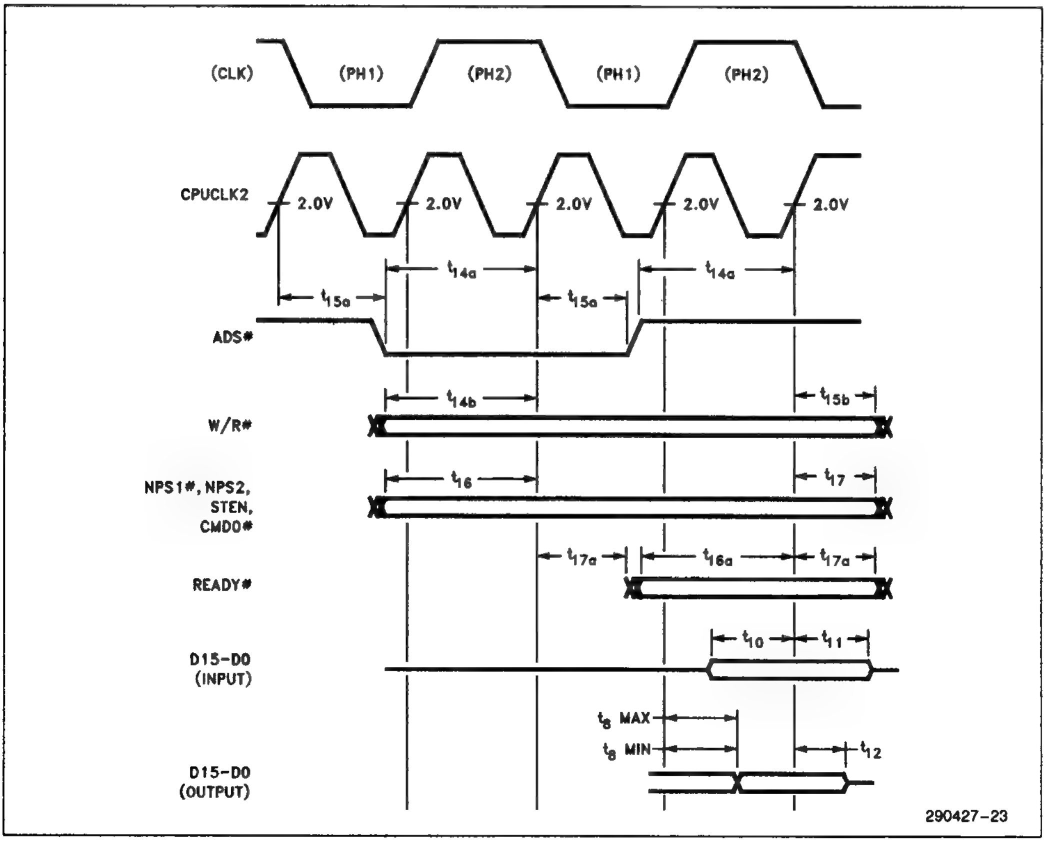
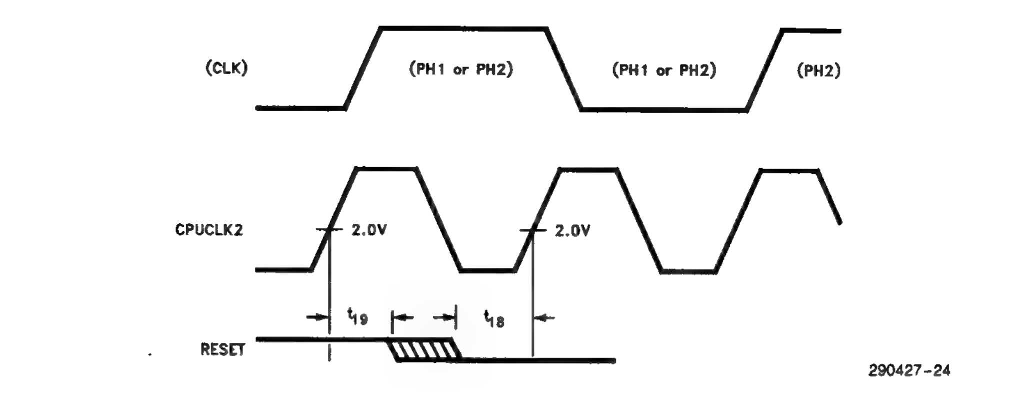


Figure 7-4. Input and I/O Signals



NOTE:
The second internal processor phase following RESET high to low transition is PH2.

Figure 7-5. RESET Signal

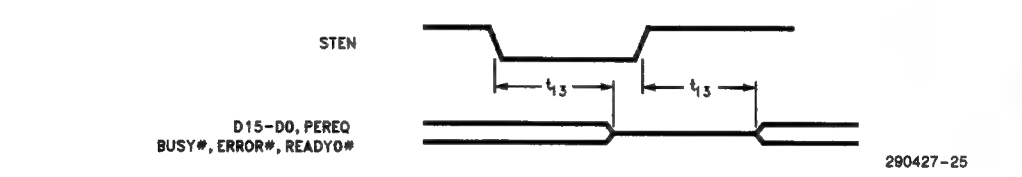
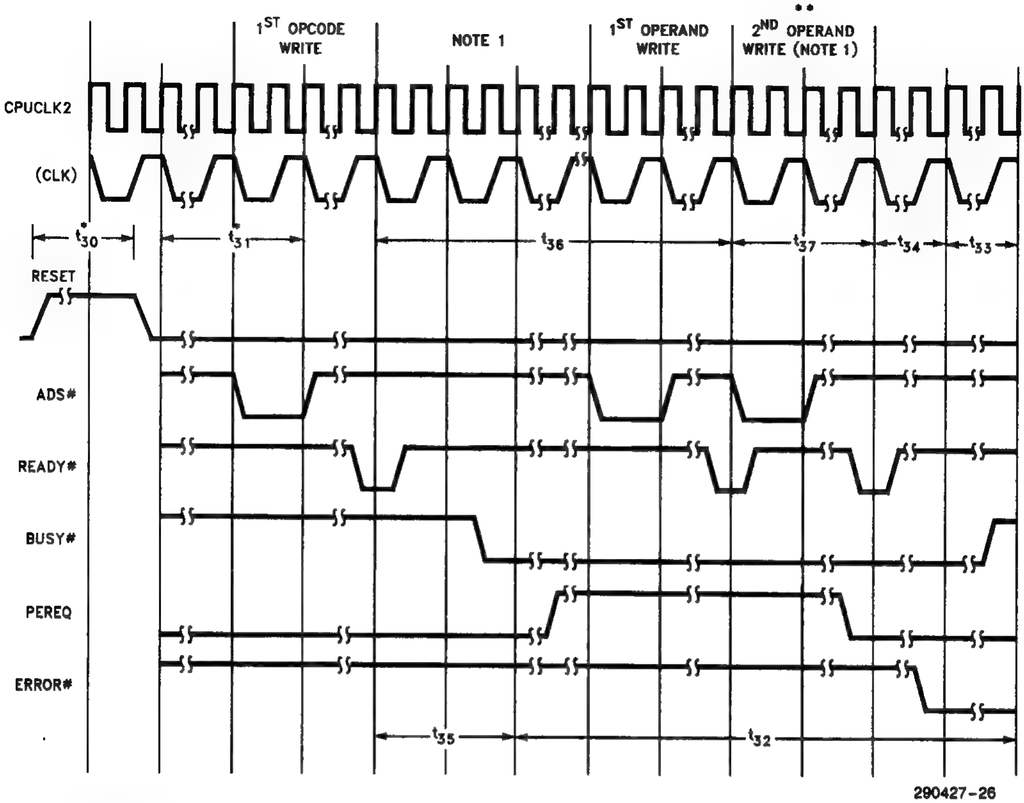


Figure 7-6. Float from STEN



*In NUMCLK2's
**or last operand

NOTE:
1. Memory read (operand) cycle is not shown.

Figure 7-7. Other Parameters

8.0 INTEL387 SL MOBILE MATH COPROCESSOR INSTRUCTION SET

Instructions for the Intel387 SL Mobile Math CoProcessor assume one of the five forms shown in Table 8-1. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B, which identifies the ESCAPE class of instruction. Instructions that refer to memory operands specify addresses using the CPU's addressing modes.

MOD (Mode field) and R/M (Register/Memory specifier) have the same interpretation as the corresponding fields of CPU instructions (refer to Programmer's Reference Manual for the CPU). SIB (Scale Index Base) byte and DISP (displacement) are optionally present in instructions that have MOD and R/M fields. Their presence depends on the values of MOD and R/M, as for instructions of the CPU.

The instruction summaries that follow in Table 8-2 assume that the instruction has been prefetched, decoded, and is ready for execution; that bus cycles do not require wait states; that there are no local bus HOLD requests delaying processor access to the bus; and that no exceptions are detected during instruction execution. If the instruction has MOD and R/M fields that call for both base and index registers, add one clock.

Table 8-1. Instruction Formats

	Instruction									Optional Fields	
	First Byte				Second Byte						
1	11011	OPA		1	MOD		1	OPB	R/M	SIB	DISP
2	11011	MF		OPA	MOD		OPB*		R/M	SIB	DISP
3	11011	d	P	OPA	1	1	OPB*		ST(i)		
4	11011	0	0	1	1	1	1	OP			
5	11011	0	1	1	1	1	1	OP			
	15–11	10	9	8	7	6	5	4 3 2 1 0			

OP = Instruction opcode, possibly split into two fields OPA and OPB
MF = Memory Format
00 - 32-bit real
01 - 32-bit integer
10 - 64-bit real
11 - 16-bit integer
d = Destination
0 - Destination is ST(0)
1 - Destination is ST(i)
R XOR d = 0 - Destination (op) Source
R XOR d = 1 - Source (op) Destination
*In FSUB and FDIV, the low-order bit of OPB is the R (reversed) bit
P = POP
0 - Do not pop stack
1 - Pop stack after operation
ESC = 11011
ST(i) = Register stack element i
000 = Stack top
001 = Second stack element
•
•
•
111 = Eighth stack element

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2–6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
DATA TRANSFER							
FLD = Load[a]							
Integer/real memory to ST(0)	ESC MF 1	MOD 000 R/M	SIB/DISP	11–20	28–44	20–27	42–53
Long integer memory to ST(0)	ESC 111	MOD 101 R/M	SIB/DISP	30–58			
Extended real memory to ST(0)	ESC 011	MOD 101 R/M	SIB/DISP	16–47			
BCD memory to ST(0)	ESC 111	MOD 100 R/M	SIB/DISP	49–101			
ST(i) to ST(0)	ESC 001	11000 ST(i)		7–12			
FST = Store							
ST(0) to integer/real memory	ESC MF 1	MOD 010 R/M	SIB/DISP	27–45	59–78	59	58–76
ST(0) to ST(i)	ESC 101	11010 ST(i)		7–11			
FSTP = Store and Pop							
ST(0) to integer/real memory	ESC MF 1	MOD 011 R/M	SIB/DISP	27–45	59–78	59	58–76
ST(0) to long integer memory	ESC 111	MOD 111 R/M	SIB/DISP	64–86			
ST(0) to extended real memory	ESC 011	MOD 111 R/M	SIB/DISP	50–56			
ST(0) to BCD memory	ESC 111	MOD 110 R/M	SIB/DISP	116–194			
ST(0) to ST(i)	ESC 101	11011 ST (i)		7–11			
FXCH = Exchange							
ST(i) and ST(0)	ESC 001	11001 ST(i)		10–17			
COMPARISON							
FCOM = Compare							
Integer/real memory to ST(0)	ESC MF 0	MOD 010 R/M	SIB/DISP	15–27	36–54	18–31	39–62
ST(i) to ST(0)	ESC 000	11010 ST(i)		13–21			
FCOMP = Compare and pop							
Integer/real memory to ST(0)	ESC MF 0	MOD 011 R/M	SIB/DISP	15–27	36–54	18–31	39–62
ST(i) to ST(0)	ESC 000	11011 ST(i)		13–21			
FCOMPP = Compare and pop twice							
ST(1) to ST(0)	ESC 110	1101 1001		13–21			
FTST = Test ST(0)	ESC 001	1110 0100		17–25			
FUCOM = Unordered compare	ESC 101	11100 ST(i)		13–21			
FUCOMP = Unordered compare and pop	ESC 101	11101 ST(i)		13–21			
FUCOMPP = Unordered compare and pop twice	ESC 010	1110 1001		13–21			
FXAM = Examine ST(0)	ESC 001	1110 0101		24-37			

Shaded areas indicate instructions not available in 8087/80287.

NOTE:
a. When loading single or double precision zero from memory, add 5 clocks.

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2–6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
ARITHMETIC							
FADD = Add							
Integer/real memory to ST(0)	ESC MF 0	MOD 000 R/M	SIB/DISP	14–31	36–58	19–38	38–64
ST(i) and ST(0)	ESC d P 0	11000 ST(i)	SIB/DISP	12–26[b]			
FSUB = Subtract							
Integer/real memory with ST(0)	ESC MF 0	MOD 10 R R/M	SIB/DISP	14–31	36–58	19–38	38–64[c]
ST(i) to ST(0)	ESC d P 0	1110 R R/M		12–26[d]			
FMUL = Multiply							
Integer/real memory with ST(0)	ESC MF 0	MOD 001 R/M	SIB/DISP	21–33	45–73	27–57	46–74
ST(i) and ST(0)	ESC d P 0	1100 1 R/M		17–50[e]			
FDIV = Divide							
Integer/real memory with ST(0)	ESC MF 0	MOD 11 R R/M	SIB/DISP	79–87	103–116[f]	85–95	105–124[g]
ST(i) and ST(0)	ESC d P 0	1111 R R/M		77–80[h]			
FSQRT[i] = Square root	ESC 001	1111 1010		97–111			
FSCALE = Scale ST(0) by ST(1)	ESC 001	1111 1101		44–82			
FPREM = Partial remainder	ESC 001	1111 1000		56–140			
FPREM1 = Partial remainder (IEEE)	ESC 001	1111 0101		81–106			
FRNDINT = Round ST(0) to integer	ESC 001	1111 1100		41–62			
FXTRACT = Extract components of ST(0)	ESC 001	1111 0100		42–63			
FABS = Absolute value of ST(0)	ESC 001	1110 0001		14–21			
FCHS = Change sign of ST(0)	ESC 001	1110 0000		17–24			
TRANSCENDENTAL							
FCOS[k] = Cosine of ST(0)	ESC 001	1111 1111		122–680			
FPTAN[k] = Partial tangent of ST(0)	ESC 001	1111 0010		162–430[j]			
FPATAN = Partial arctangent of ST(0)	ESC 001	1111 0011		250–420			
FSIN[k] = Sine of ST(0)	ESC 001	1111 1110		121–680			
FSINCOS[k] = Sine and cosine of ST(0)	ESC 001	1111 1011		150–650			
F2XM1[l] = $2^{ST(0)} - 1$	ESC 001	1111 0000		167–410			
FYL2X[m] = $ST(1) * \log_2 ST(0)$	ESC 001	1111 0001		99–436			
FYL2XP1[n] = $ST(1) * \log_2[ST(0) + 1.0]$	ESC 001	1111 1001		210–447			

Shaded areas indicate instructions not available in 8087/80287.

NOTES:

b. Add 3 clocks to the range when d = 1.
c. Add 1 clock to each range when R = 1.
d. Add 3 clocks to the range when d = 0.
e. typical = 52 (When d = 0, 46–54, typical = 49).
f. Add 1 clock to the range when R = 1.
g. 135–141 when R = 1.
h. Add 3 clocks to the range when d = 1.
i. $-0 \leq ST(0) \leq +\infty$.
j. These timings hold for operands in the range $|x| < \pi$. For operands not in this range, up to 76 additional clocks may be needed to reduce the operand.
k. $0 \leq |ST(0)| < 2^{63}$.
l. $-1.0 \leq ST(0) \leq 1.0$.
m. $0 \leq ST(0) < \infty$, $-\infty < ST(1) < +\infty$.
n. $0 \leq |ST(0)| < [2\text{-SQRT}(2)]/2$, $-\infty < ST(1) < +\infty$.

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2–6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
CONSTANTS							
FLDZ = Load +0.0 to ST(0)	ESC 001	1110 1110		10–17			
FLD1 = Load +1.0 to ST(0)	ESC 001	1110 1000		15–22			
FLDPI = Load π to ST(0)	ESC 001	1110 1011		26–36			
FLDL2T = Load log$_2$(10) to ST(0)	ESC 001	1110 1001		26–36			
FLDL2E = Load log$_2$(e) to ST(0)	ESC 001	1110 1010		26–36			
FLDLG2 = Load log$_{10}$(2) to ST(0)	ESC 001	1110 1100		25–35			
FLDLN2 = Load log$_e$(2) to ST(0)	ESC 001	1110 1101		26–38			
PROCESSOR CONTROL							
FINIT = Initialize Math CoProcessor	ESC 011	1110 0011		33			
FLDCW = Load control word from memory	ESC 001	MOD 101 R/M	SIB/DISP	19			
FSTCW = Store control word to memory	ESC 001	MOD 111 R/M	SIB/DISP	15			
FSTSW = Store status word to memory	ESC 101	MOD 111 R/M	SIB/DISP	15			
FSTSW AX = Store status word to AX	ESC 111	1110 0000		13			
FSTDW AX = Store device word to AX	ESC 111	1110 0001		13			
FSTSG AX = Store signature word to AX	ESC 111	1110 0010		13			
FCLEX = Clear exceptions	ESC 011	1110 0010		11			
FSTENV = Store environment	ESC 001	MOD 110 R/M	SIB/DISP	117–118			
FLDENV = Load environment	ESC 001	MOD 100 R/M	SIB/DISP	85			
FSAVE = Save state	ESC 101	MOD 110 R/M	SIB/DISP	402–403			
FRSTOR = Restore state	ESC 101	MOD 100 R/M	SIB/DISP	415			
FINCSTP = Increment stack pointer	ESC 001	1111 0111		21			
FDECSTP = Decrement stack pointer	ESC 001	1111 0110		22			
FFREE = Free ST(i)	ESC 101	1100 0 ST(i)		18			
FNOP = No operations	ESC 001	1101 0000		12			

Shaded areas indicate instructions not available in 8087/80287.

9.0 REVISION HISTORY

This Intel387 SL Mobile Math CoProcessor data sheet, version-001, is the original version. Future updates and improvements will be summarized in this section for your convenience.

APPENDIX A
PROGRAMMER'S REFERENCE MANUAL UPDATES

Appendix A contains the Intel387 Math CoProcessor Programmer's Reference Manual Updates for the new Device Word Register and Signature Word Register. These registers are only available on the Intel387 SL Mobile Math CoProcessor at this time.

FSTDW/FNSTDW—Store Device Word Register

OpCode	Instruction	Clocks	Description
9B DF E1	FSTDW AX	13+ at least 6 for FWAIT	Store i387 SL Mobile device word to AX register after checking for unmasked floating-point errors.
DF E1	FNSTDW AX	13	Store i387 SL Mobile device word to AX register without checking for unmasked floating-point errors.

Operation

DEST <- DW;

Description

FSTDW and FNSTDW write the value of the Intel387 SL Mobile Math CoProcessor device word to the AX register.

FPU Flags Affected

None

Numeric Exceptions

None

Protected Mode Exceptions

#NM if either EM or TS in CR0 is set.

Real Address Mode Exceptions

Interrupt 7 if either EM or TS in CR0 is set.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode

NOTES:

FSTDW checks for unmasked floating-point error conditions before storing the device word; FNSTDW does not.

FSTDW and FNSTDW are used for feature recognition and should be used after FINIT and before the status word has changed. On devices without the device word, the status word will be transferred. Execution of the instruction after the status word has changed can produce indeterminate results.

When FNSTDW AX is executed, the AX register is updated before the CPU processor executes any further instruction.

FSTSG/FNSTSG—Store Signature Register

OpCode	Instruction	Clocks	Description
9B DF E2	FSTSG AX	13+ at least 6 for FWAIT	Store i387 SL Mobile signature to AX register after checking for unmasked floating-point errors.
DF E2	FNSTSG AX	13	Store i387 SL Mobile signature to AX register without checking for unmasked floating-point errors.

Operation

DEST <- Signature;

Description

FSTSG and FNSTSG write the value of the Intel387 SL Mobile Math CoProcessor signature register to the AX register.

FPU Flags Affected

None

Numeric Exceptions

None

Protected Mode Exceptions

#NM if either EM or TS in CR0 is set.

Real Address Mode Exceptions

Interrupt 7 if either EM or TS in CR0 is set.

Virtual 8086 Mode Exceptions

Same exceptions as in Real Address Mode

NOTES:

FSTSG checks for unmasked floating-point error conditions before storing the signature word; FNSTSG does not.

FSTSG and FNSTSG are used for device and stepping recognition and should be used after FINIT and before the status word has changed. On devices without the signature word, the status word will be transferred. Execution of the instruction after the status word has changed can produce indeterminate results.

When FNSTSG AX is executed, the AX register is updated before the CPU processor executes any further instructions.

APPENDIX B INTEL387 SL MOBILE MATH COPROCESSOR COMPATIBILITY

B.1 Intel387 SX Compatibility

This section summarizes the differences between the Intel387 SL Mobile Math CoProcessor and the Intel387 SX Math CoProcessor. The Intel387 SL Mobile Math CoProcessor is fully compatible with the Intel387 SX Math CoProcessor and will execute Intel386 CPU/Intel387 Math CoProcessor programs without modification.

1. The Intel387 SL Mobile Math CoProcessor implements a Device Word register, for static recognition, which did not exist in the Intel387 SX Math CoProcessor.
2. The Intel387 SL Mobile Math CoProcessor includes a Signature register for device stepping information.
3. Due to increased performance, diagnostic programs must not rely upon instruction execution speeds for test purposes.

B.2 8087/80287 Compatibility

This section summarizes the differences between the Intel387 SL Mobile Math CoProcessor and the 80287 Math CoProcessor. (This does not include the Intel287 XL Math CoProcessor which is compatible with the Intel387 SX Math CoProcessor.) Any migration from the 8087 directly to the Intel387 SL Mobile Math CoProcessor must also take into account the differences between the 8087 and the 80287 Math CoProcessor as listed in Appendix C.

Many changes have been designed into the Intel387 SL Mobile Math CoProcessor to directly support the IEEE standard in hardware. These changes result in increased performance by eliminating the need for software that supports the standard.

B.2.1 GENERAL DIFFERENCES

The Intel387 SL Mobile Math CoProcessor supports only affine closure for infinity arithmetic, not projective closure.

Operands for FSCALE and FPATAN are no longer restricted in range (except for ± ∞); F2XM1 and FPTAN accept a wider range of operands.

Rounding control is in effect for FLD constant.

Software cannot change entries of the tag word to values (other than empty) that differ from actual register contents.

After reset, FINIT, and incomplete FPREM, the Intel387 SL Mobile Math CoProcessor resets to zero the condition code bits C_3-C_0 of the status word.

In conformance with the IEEE standard, the Intel387 SL Mobile Math CoProcessor does not support the special data formats pseudo-zero, pseudo-NaN, pseudo-infinity, and unnormal.

The denormal exception has a different purpose on the Intel387 SL Mobile Math CoProcessor. A system that uses the denormal exception handler solely to normalize the denormal operands, would better mask the denormal exception on the Intel387 SL Math CoProcessor. The Intel387 SL Mobile Math CoProcessor automatically normalizes denormal operands when the denormal exception is masked.

B.2.2. EXCEPTIONS

A number of differences exist due to changes in the IEEE standard and to functional improvements to the architecture of the Intel387 SL Mobile Math CoProcessor:

1. When the overflow or underflow exception is masked, the Intel387 SL Mobile Math CoProcessor differs from the 80287 in rounding when overflow or underflow occurs. The Intel387 SL Mobile Math CoProcessor produces results that are consistent with the rounding mode.
2. When the underflow exception is masked, the Intel387 SL Mobile Math CoProcessor sets its underflow flag only if there is also a loss of accuracy during denormalization.
3. Fewer invalid-operations exceptions due to denormal operand, because the instructions FSQRT, FDIV, FPREM, and conversions to BCD or to integer normalize denormal operands before proceeding.
4. The FSQRT, FBSTP, and FPREM instructions may cause underflow, because they support denormal operands.
5. The denormal exception can occur during the transcendental instruction and the FXTRACT instruction.
6. The denormal exception no longer takes precedence over all other exceptions.
7. When the denormal exception is masked, the Intel387 SL Mobile Math CoProcessor automatically normalizes denormal operands. The 8087/80287 performs unnormal arithmetic, which might produce an unnormal result.
8. When the operand is zero, the FXTRACT instruction reports a zero-divide exception and leaves $-\infty$ in ST(1).
9. The status word has a new bit (SF) that signals when invalid-operation exceptions are due to stack underflow or overflow.
10. FLD *extended precision* no longer reports denormal exceptions, because the instruction is not numeric.
11. FLD *single/double precision* when the operand is denormal converts the number to extended precision and signals the denormal operand exception. When loading a signaling NaN, FLD *single/double precision* signals an invalid-operation exception.
12. The Intel387 SL Mobile Math CoProcessor only generates quiet NaNs (as on the 80287); however, the Intel387 SL Mobile Math CoProcessor distinguishes between quiet NaNs and signaling NaNs. Signaling NaNs trigger exceptions when they are used as operands; quiet NaNs do not (except for FCOM, FIST, and FBSTP which also raise IE for quiet NaNs).
13. When stack overflow occurs during FPTAN and overflow is masked, both ST(0) and ST(1) contain quiet NaNs. The 80287/8087 leaves the original operand in ST(1) intact.
14. When the scaling factor is $\pm\infty$, the FSCALE instruction behaves as follows:
    - FSCALE (0, ∞) generates the invalid operation exception.
    - FSCALE (finite, $-\infty$) generates zero with the same sign as the scaled operand.
    - FSCALE (finite, $+\infty$) generates ∞ with the same sign as the scaled operand.

    The 8087/80287 returns zero in the first case and raises the invalid-operation exception in the other cases.
15. The Intel387 SL Mobile Math CoProcessor returns signed infinity/zero as the unmasked response to massive overflow/underflow. The 8087 and 80287 support a limited range for the scaling factor; within this range either massive overflow/underflow do not occur or undefined results are produced.

APPENDIX C
COMPATIBILITY BETWEEN THE 80287 AND 8087 MATH COPROCESSOR

The 80286/80287 operating in Real Address mode will execute 8086/8087 programs without major modification. However, because of differences in the handling of numeric exceptions by the 80287 Math CoProcessor and the 8087 Math CoProcessor, exception handling routines *may* need to be changed. This appendix summarizes the differences between the 80287 Math CoProcessor and the 8087 Math CoProcessor, and provides details showing how 8087/8087 programs can be ported to the 80286/80287.

1. The Math CoProcessor signals exceptions through a dedicated ERROR# line to the 80286. The Math CoProcessor error signal does not pass through an interrupt controller (the 8087 INT signal does). Therefore, any interrupt controller oriented instructions in numeric exception handlers for the 8086/8087 should be deleted.
2. The 8087 instructions FENI and FDISI perform no useful function in the 80287. If the 80287 encounters one of these opcodes in its instruction stream, the instruction will effectively be ignored; none of the 80287 internal states will be updated. While 8086/8087 programs containing the instruction may be executed on the 80286/80287, it is unlikely that the exception handling routines containing these instructions will be completely portable to the 80287.
3. Interrupt vector 16 must point to the numeric exception handling routine.
4. The ESC instruction address saved in the 80287 includes any leading prefixes before the ESC opcode. The corresponding address saved in the 8087 does not include leading prefixes.
5. In Protected Address mode, the format of the 80287's saved instruction and address pointers is different than for the 8087. The instruction opcode is not saved in Protected mode; exception handlers will have to retrieve the opcode from memory if needed.
6. Interrupt 7 will occur in the 80286 when executing ESC instructions with either TS (task switched) or EM (emulation) of the 80286 MSW set (TS = 1 or EM = 1). It TS is set, then a WAIT instruction will also cause interrupt 7. An exception handler should be included in 80286/80287 code to handle these situations.
7. Interrupt 9 will occur if the second or subsequent words of a floating point operand fall outside a segment's size. Interrupt 13 will occur if the starting address of a numeric operand falls outside a segment's size. An exception handler should be included in 80286/80287 code to report these programming errors.
8. Except for the processor control instructions, all of the 80287 numeric instructions are automatically synchronized by the 80286 CPU; the 80286 CPU automatically tests the BUSY# line from the 80287 to ensure that the 80287 has completed its previous instruction before executing the next ESC instruction. No explicit WAIT instructions are required to assure this synchronization. For the 8087 used witth 8086 and 8088 processors, explicit WAITs are required before each numeric instruction to ensure synchronization. Although 8086/8087 programs having explicit WAIT instructions will execute perfectly on the 80286/80287 without reassembly, these WAIT instructions are unnecessary.
9. Since the 80287 does not require WAIT instructions before each numeric instruction, the ASM286 assembler does not automatically generate these WAIT instuctions. The ASM86 assembler, however, automatically precedes every ESC instruction with a WAIT instruction. Although numeric routines generated using the ASM86 assembler will generally execute correctly on the 80286/80287, reassembly using ASM286 may result in a more compact code image.

The processor control instructions for the 80287 may be coded using either a WAIT or No-WAIT form of mnemonic. The WAIT forms of these instructions cause ASM286 to precede the ESC instructions with a CPU WAIT instruction, in the identical manner as does ASM86.